

# Reactive Synthesis using MTBDDs

Alexandre Duret-Lutz   Dror Fried  
Giuseppe De Giacomo   Lucas M. Tabajara  
Marcin Jurdzinski   Moshe Y. Vardi   Nadav Alon  
Nir Piterman   Shufang Zhu   Supratik Chakraborty

# Outline

## 1 $LTL_f$ Synthesis

- Introduction
- MTBDD-based Approach
- MTDFA as Game
- From  $LTL_f$  to MTDFA
- On-the-fly Construction
- Benchmark
- Conclusion

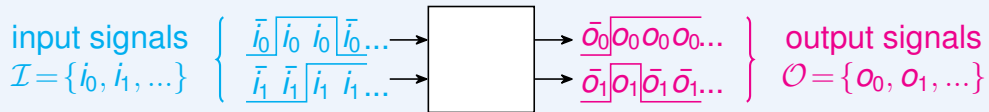
## 2 Obligation Synthesis (LTL)

## 3 Synthesis Under Partial Observability



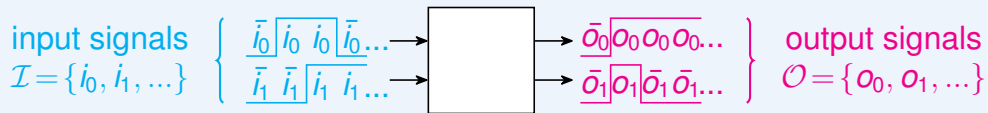
# Reactive Synthesis in a Nutshell

A reactive controller produces output as a reaction to its input



# Reactive Synthesis in a Nutshell

A **reactive controller** produces output as a reaction to its input



## The reactive synthesis problems

Given a specification relating **input signals** and **output signals** over time:

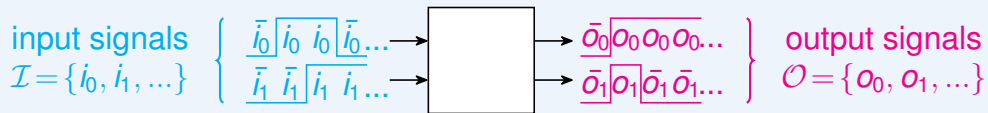
**Realizability:** decide if a controller exists;

**Synthesis:** construct it (e.g., as an And-Inverter Graph).

our focus

# Reactive Synthesis in a Nutshell

A reactive controller produces output as a reaction to its input



## The reactive synthesis problems

Given a specification relating **input signals** and **output signals** over time:

**Realizability:** decide if a controller exists;

our focus

**Synthesis:** construct it (e.g., as an And-Inverter Graph).

## Semantics for an $LTL_f$ specification

Any execution of the controller, seen as an infinite word such as

“ $\bar{i}_0 \bar{i}_1 \bar{o}_0 \bar{o}_1; i_0 \bar{i}_1 o_0 o_1; i_0 i_1 o_0 \bar{o}_1; \dots$ ”, must have a finite prefix satisfying the specification.

# Text-Book Approach

1. LTL<sub>f</sub> specification  $\varphi$

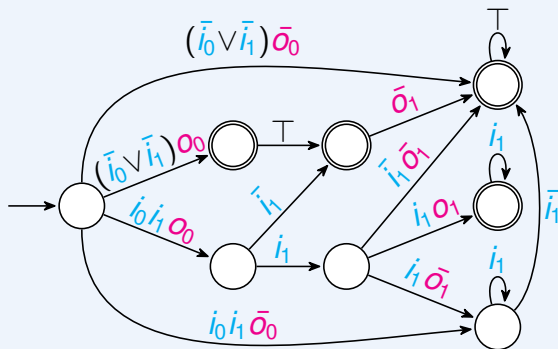
$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X!X! o_1)$$

# Text-Book Approach

## 1. LTL<sub>f</sub> specification $\varphi$

$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## 2. Build a DFA $\mathcal{A}_\varphi$

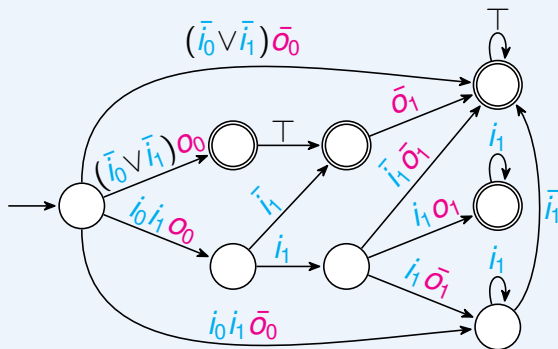


# Text-Book Approach

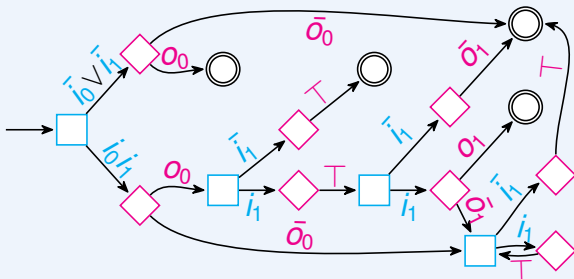
## 1. LTL<sub>f</sub> specification $\varphi$

$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## 2. Build a DFA $\mathcal{A}_\varphi$



## 3. Make it a Reachability Game

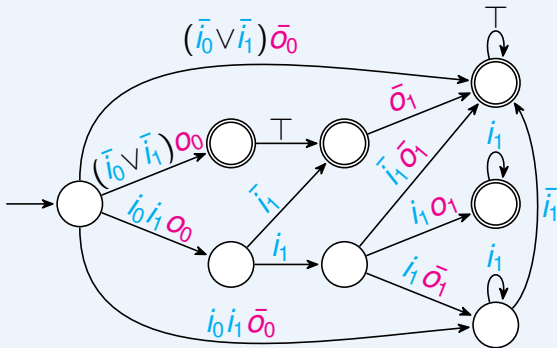


## Text-Book Approach

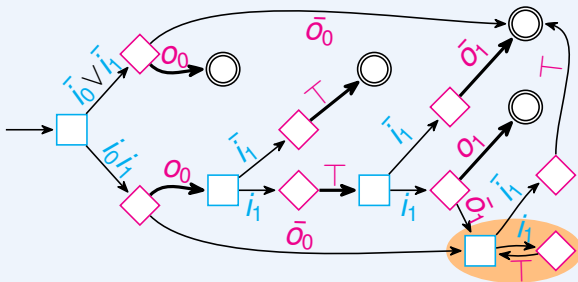
## 1. LTL<sub>f</sub> specification $\varphi$

$$(i_0 \wedge \mathbf{G}i_1) \leftrightarrow (o_0 \wedge \mathbf{X}! \mathbf{X}! o_1)$$

## 2. Build a DFA $\mathcal{A}_\varphi$



### 3. Make it a Reachability Game



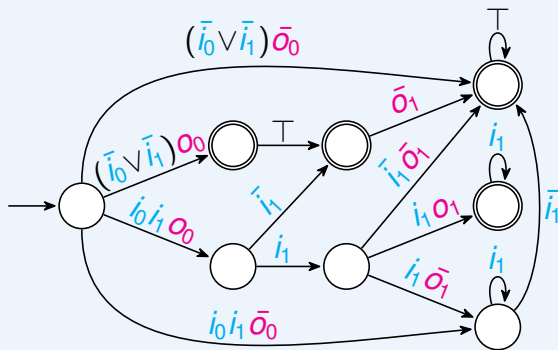
## 4. Solve it

# Text-Book Approach

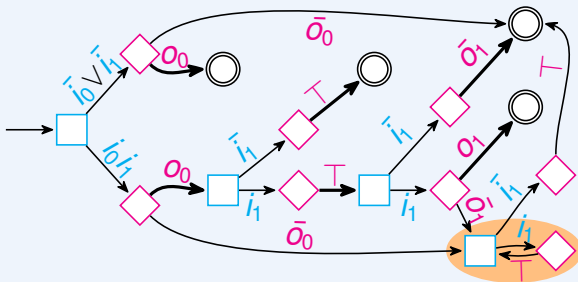
## 1. LTL<sub>f</sub> specification $\varphi$

$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## 2. Build a DFA $\mathcal{A}_\varphi$

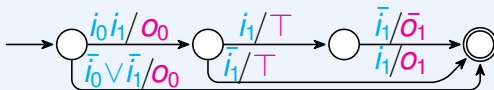


## 3. Make it a Reachability Game



## 4. Solve it

## 5. Extract a Controller if Desired

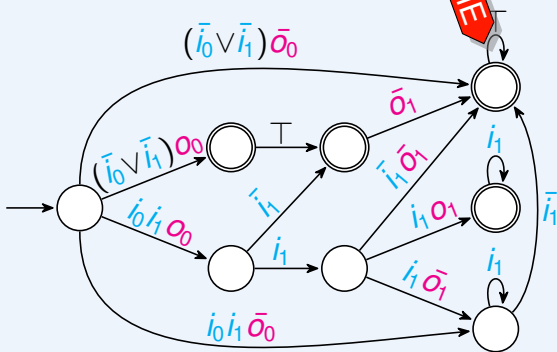


## Text-Book Approach

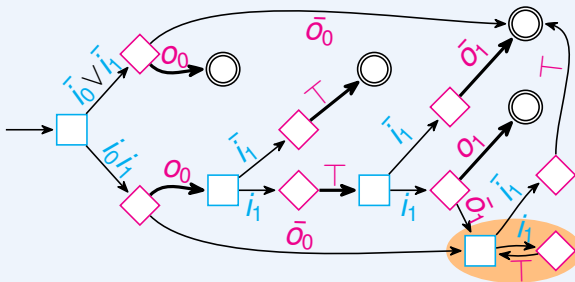
## 1. LTL<sub>f</sub> specification $\varphi$

$$(i_0 \wedge \mathbf{G}i_1) \leftrightarrow (o_0 \wedge \mathbf{X}! \mathbf{X}! o_1)$$

## 2. Build a DFA $\mathcal{A}_\varphi$

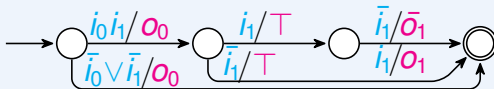


### 3. Make it a Reachability Game



## 4. Solve it

## 5. Extract a Controller if Desired

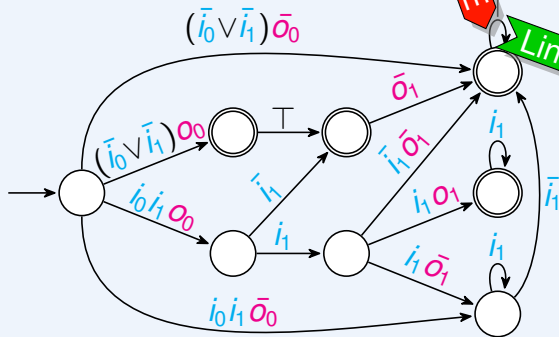


# Text-Book Approach

1. LTL<sub>f</sub> specification  $\varphi$

$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

2. Build a DFA  $\mathcal{A}_\varphi$

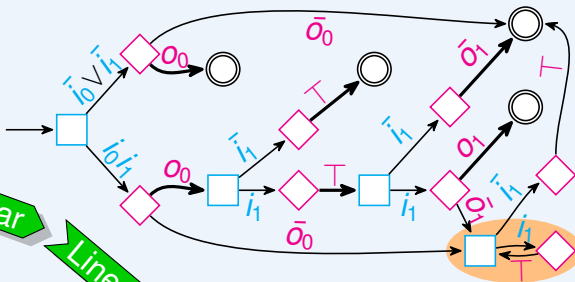


2-EXPTIME

Linear

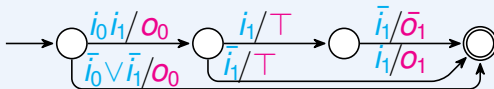
Linear

3. Make it a Reachability Game



4. Solve it

5. Extract a Controller if Desired

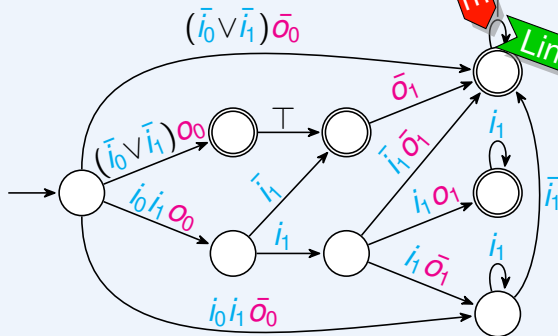


# Text-Book Approach

1. LTL<sub>f</sub> specification  $\varphi$

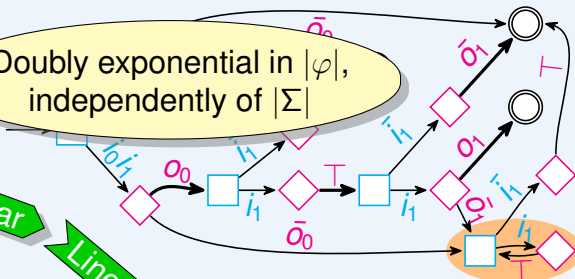
$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

2. Build a DFA  $\mathcal{A}_\varphi$



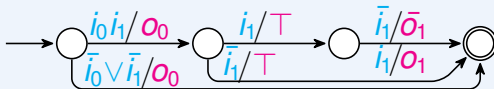
3. Make it a Reachability Game

Doubly exponential in  $|\varphi|$ ,  
independently of  $|\Sigma|$



4. Solve it

5. Extract a Controller if Desired

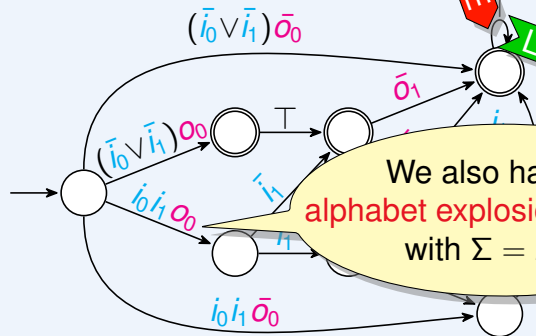


# Text-Book Approach

1. LTL<sub>f</sub> specification  $\varphi$

$$(i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X!X! o_1)$$

2. Build a DFA  $\mathcal{A}_\varphi$



2-EXPTIME

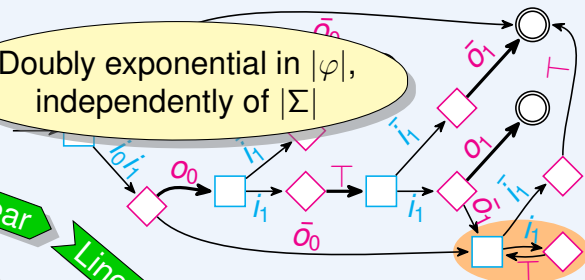
Doubly exponential in  $|\varphi|$ ,  
independently of  $|\Sigma|$

Linear

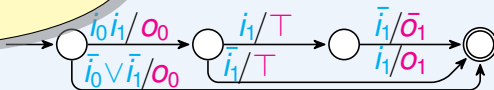
Linear

We also have an  
alphabet explosion problem,  
with  $\Sigma = 2^{I \cup O}$

3. Make it a Reachability Game



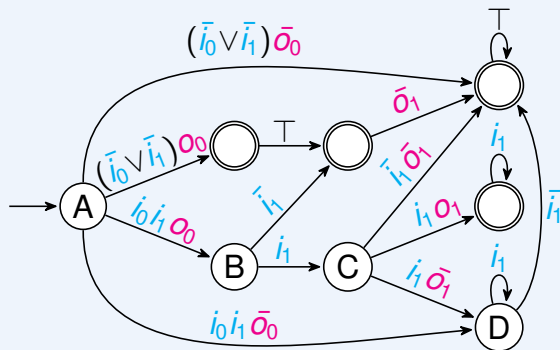
Construct a Controller if Desired



# Stopping the DFA Construction on Final States

The goal is to reach final states: we do not care about what follows.

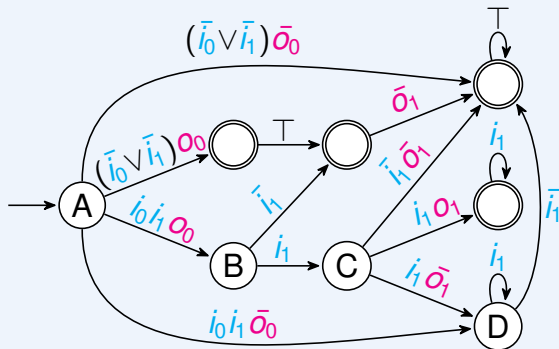
## Overkill



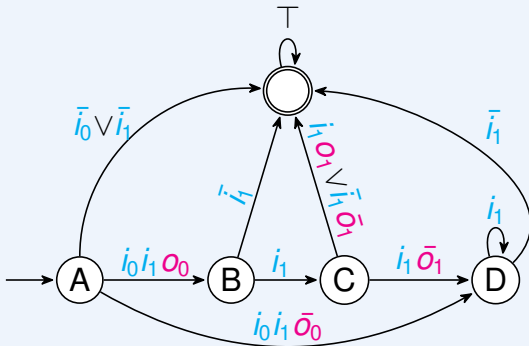
# Stopping the DFA Construction on Final States

The goal is to reach final states: we do not care about what follows.

## Overkill

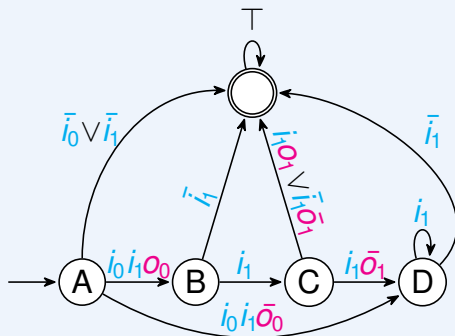


## Better



# Fighting Alphabet Explosion with MTBDDs

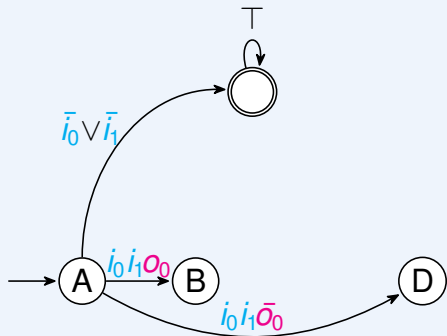
## Explicit representation



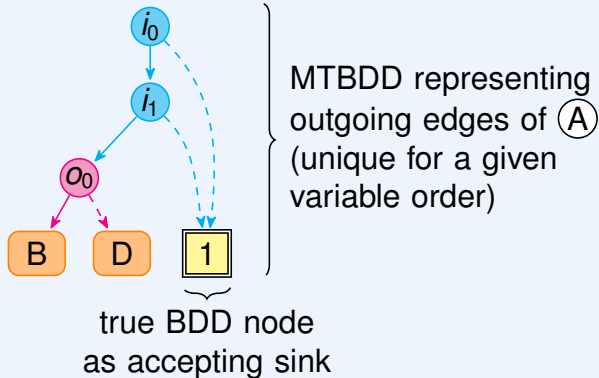
## Semi-symbolic representation: MTBDD

# Fighting Alphabet Explosion with MTBDDs

## Explicit representation

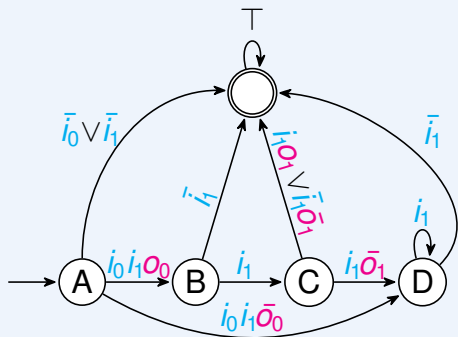


## Semi-symbolic representation: MTBDD

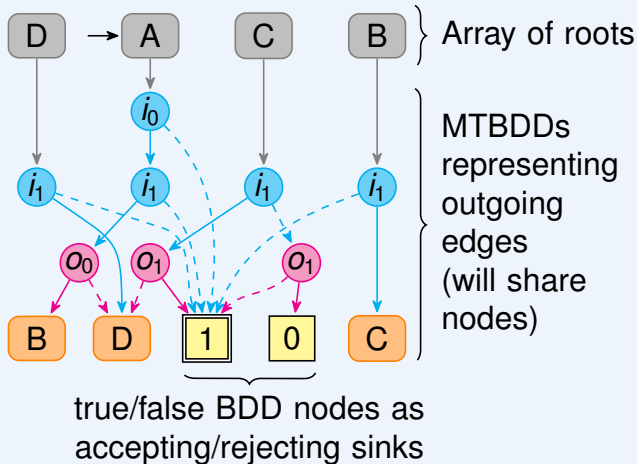


# Fighting Alphabet Explosion with MTBDDs

## Explicit representation



## Semi-symbolic representation: **MTDFA**



# LTL<sub>f</sub> Synthesis with MTDFA

## What Other Tools Have Tried

### MTDFA Constructions:

- ▶ Transform LTL<sub>f</sub>  $\rightarrow$  FOL, then use Mona for FOL  $\rightarrow$  MTDFA.
- ▶ Use Mona's MTDFA library to translate LTL<sub>f</sub> to MTDFA by composition.

**Game Solving:** convert MTDFA to BDD, and solve symbolically.

**Also exist on-the-fly approaches** that do not go through MTDFA.

## What we Suggest

- 1 Direct translation from LTL<sub>f</sub> to MTBDD, building the MTDFA one state at a time.
- 2 Solving the game on the MTDFA directly.
- 3 Doing those on-the-fly.

# LTL<sub>f</sub> Synthesis with MTDFA

## What Other Tools Have Tried

### MTDFA Constructions:

- ▶ Transform LTL<sub>f</sub>  $\rightarrow$  FOL, then use Mona for FOL  $\rightarrow$  MTDFA.
- ▶ Use Mona's MTDFA library to translate LTL<sub>f</sub> to MTDFA by composition.

**Game Solving:** convert MTDFA to BDD, and solve symbolically.

**Also exist on-the-fly approaches** that do not go through MTDFA.

## What we Suggest

- 1 Direct translation from LTL<sub>f</sub> to MTBDD, building the MTDFA one state at a time.
- 2 Solving the game on the MTDFA directly.
- 3 Doing those on-the-fly.

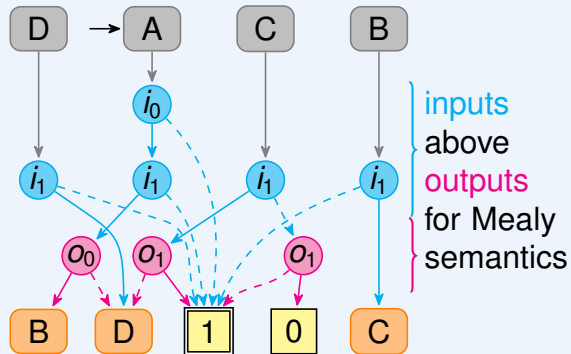
next slide

# Seeing the MTDFA as a Game Arena

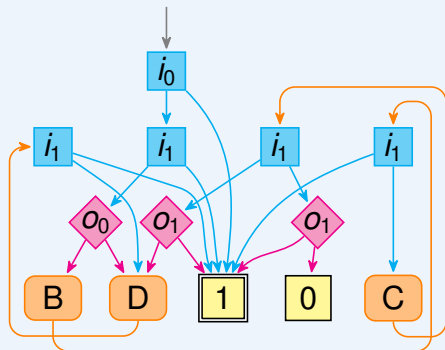
Turn input/output nodes into universal/existential vertices.

Order input/output variables according to the desired semantics (Moore/Mealy).

## The MTDFA



## The Game Interpretation

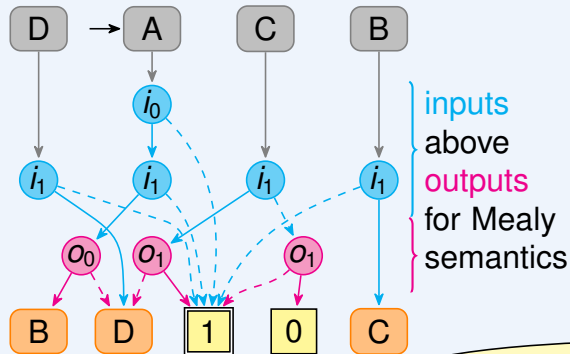


# Seeing the MTDFA as a Game Arena

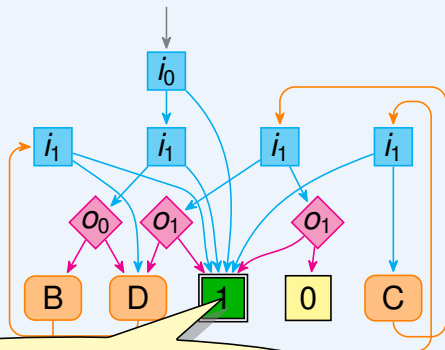
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation



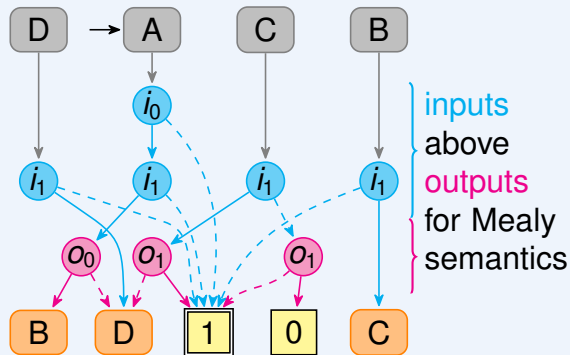
Solve by backpropagation from 1.

# Seeing the MTDFA as a Game Arena

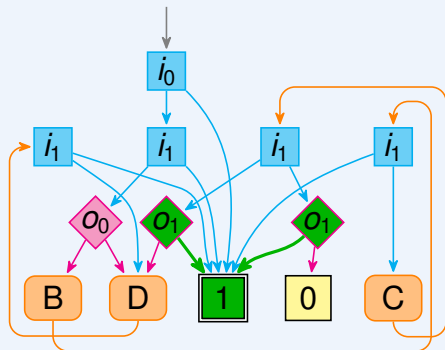
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

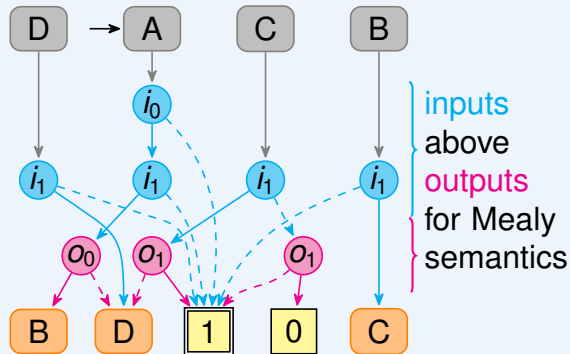


# Seeing the MTDFA as a Game Arena

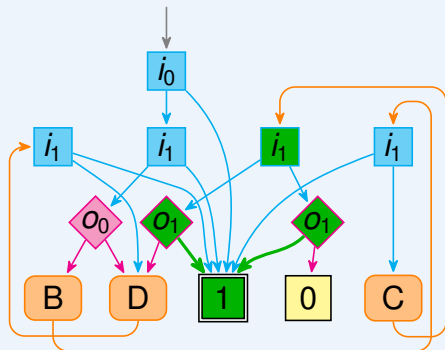
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

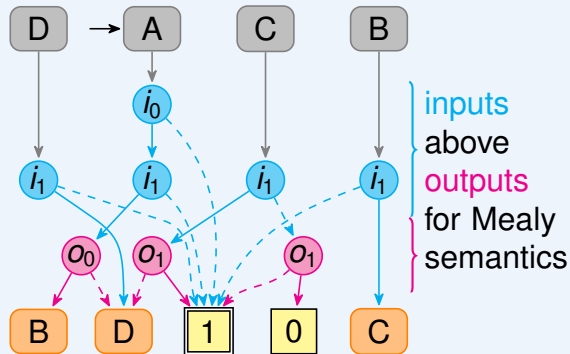


# Seeing the MTDFA as a Game Arena

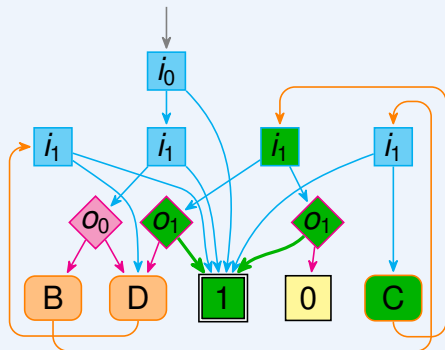
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n)

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

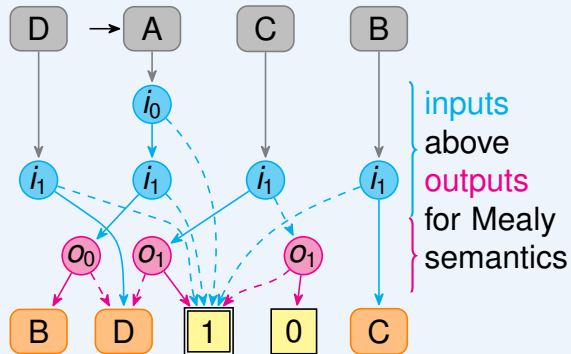


# Seeing the MTDFA as a Game Arena

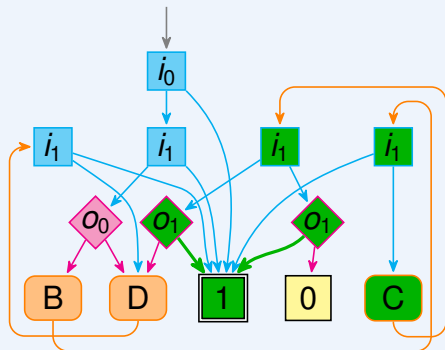
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n)

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

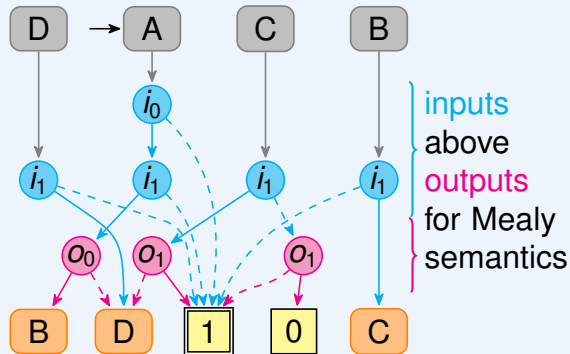


# Seeing the MTDFA as a Game Arena

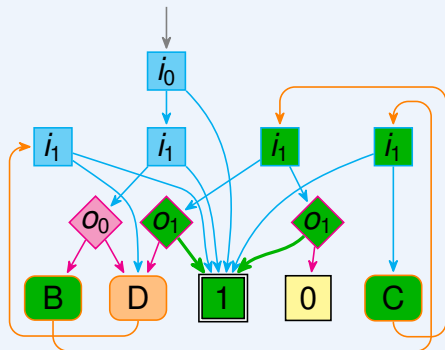
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

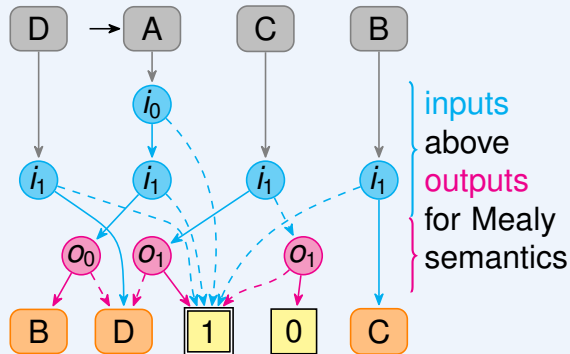


# Seeing the MTDFA as a Game Arena

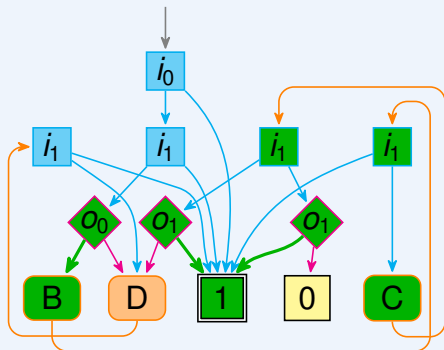
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

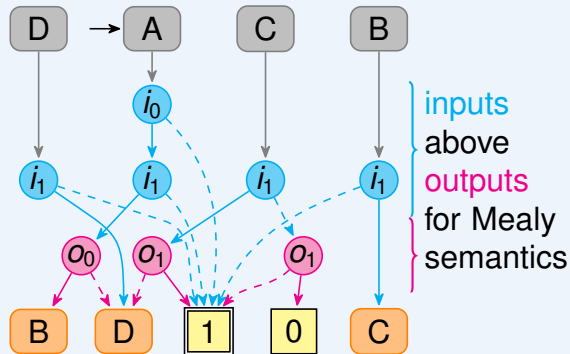


# Seeing the MTDFA as a Game Arena

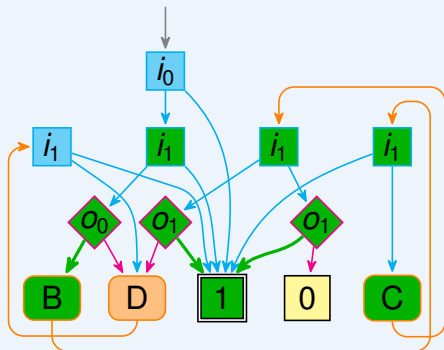
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (n

Actually stores  
reversed edges.

## The MTDFA



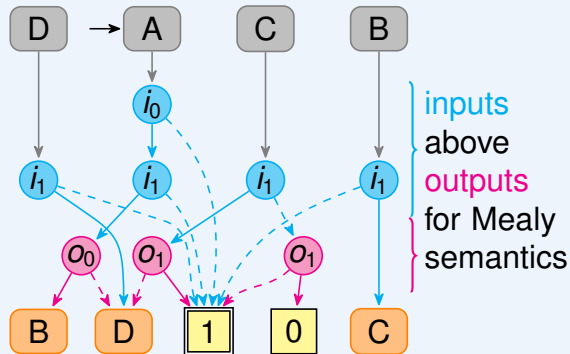
## The Game Interpretation



# Seeing the MTDFA as a Game Arena

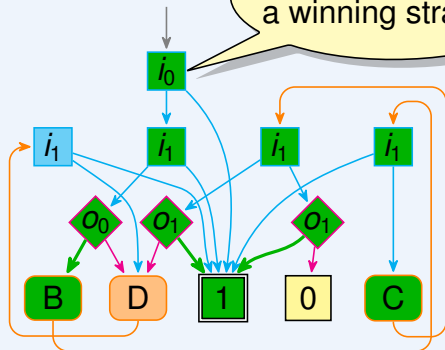
Turn input/output nodes into universal/existential vertices.  
Order input/output variables according to the desired semantics (Moore/Mealy).

## The MTDFA



## The Game Int

Output player has a winning strategy.



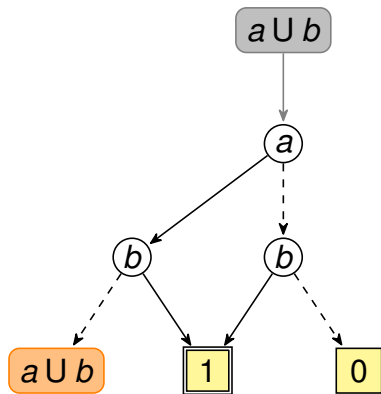
## What we Suggest

- ① Direct translation from LTL<sub>f</sub> to MTBDD, building the MTDFA one state at a time.  up next!
- ② Solving the game on the MTDFA directly. ✓
- ③ Doing those on-the-fly.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$



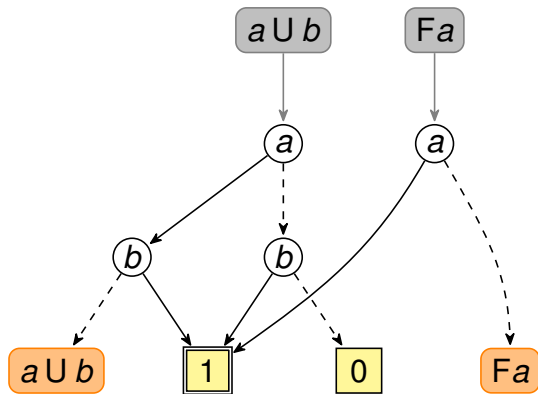
This is a deterministic representation of the *next normal form* (XNF):

$$a \cup b \equiv b \vee (a \wedge X^!(a \cup b))$$

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .



This is a deterministic representation of the *next normal form* (XNF):

$$a \cup b \equiv b \vee (a \wedge X^!(a \cup b))$$

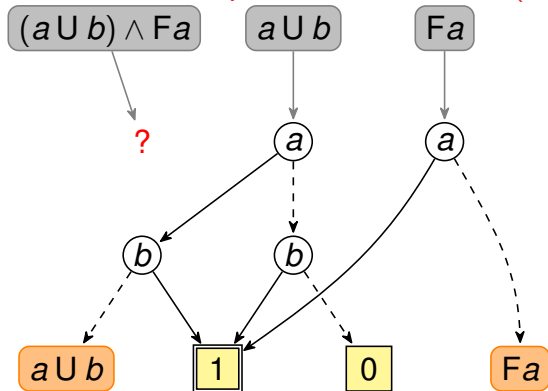
$$Fa \equiv a \vee X^!Fa$$

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :

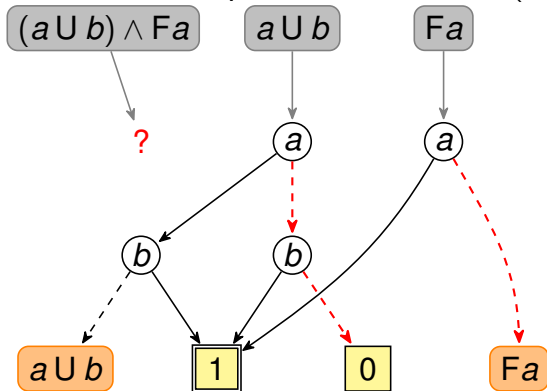


# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



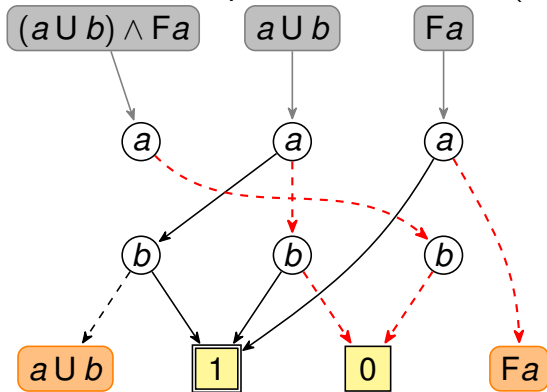
Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



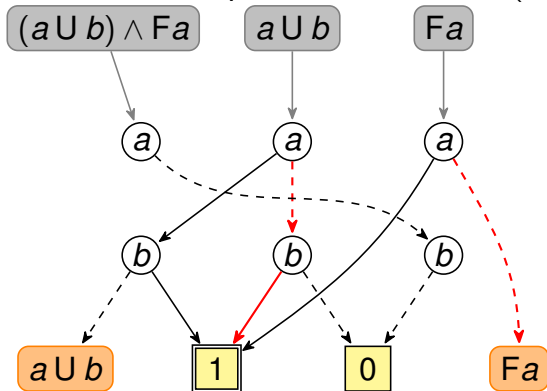
Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



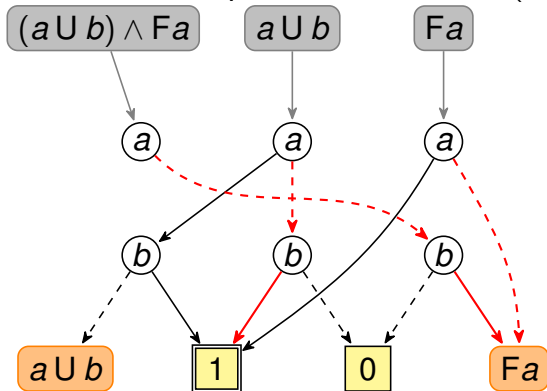
Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



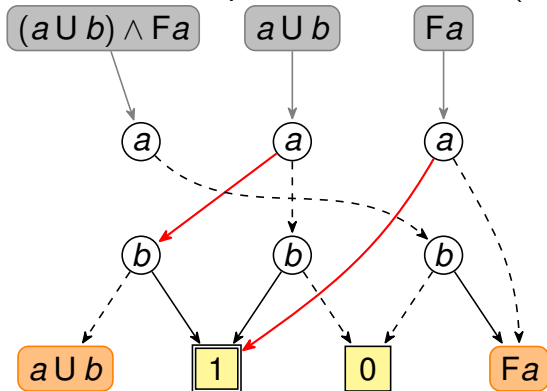
Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.

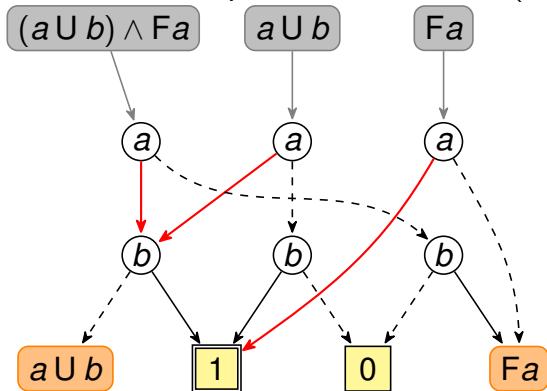
Leaves  $0$  and  $1$  can help shortcut the recursion.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



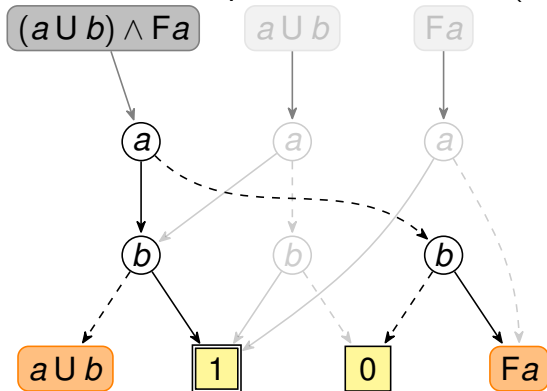
Classical BDD apply procedure, but combine terminals with “ $\wedge$ ”.  
Leaves  $0$  and  $1$  can help shortcut the recursion.

# From $LTL_f$ to MTBDD (not yet MTDFA)

Use  $LTL_f$  formulas as terminals.

Assume we know an MTBDD for the successors of  $a \cup b$ , and another for  $Fa$ .

We want to compute an MTBDD for  $(a \cup b) \wedge Fa$ :



MTBDDs for subformulas are cached (i.e., not thrown away) in case they are needed later during the construction.

# From $LTL_f$ to MTBDD: Formal Definition

$$\text{tr}(\text{ff}) = \boxed{0}$$

$$\text{tr}(\text{tt}) = \boxed{\boxed{1}}$$

$$\text{tr}(p) = \begin{array}{c} \textcircled{p} \\ \swarrow \quad \searrow \\ \boxed{0} \quad \boxed{\boxed{1}} \end{array} \quad \text{for } p \in \mathcal{I} \cup \mathcal{O}$$

$LTL_f$  operator

$$\text{tr}(X\alpha) = \boxed{\alpha}$$

$$\text{tr}(X^!\alpha) = \boxed{\alpha}$$

$$\text{tr}(\neg\alpha) = \neg \text{tr}(\alpha)$$

MTBDD operator

$$\text{tr}(\alpha \odot \beta) = \text{tr}(\alpha) \odot \text{tr}(\beta) \text{ for any } \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$$

previous slide

$$\text{tr}(\alpha \mathbf{U} \beta) = \text{tr}(\beta) \vee (\text{tr}(\alpha) \wedge \boxed{\alpha \mathbf{U} \beta})$$

$$\text{tr}(F\alpha) = \text{tr}(\alpha) \vee \boxed{F\alpha}$$

$$\text{tr}(\alpha \mathbf{R} \beta) = \text{tr}(\beta) \wedge (\text{tr}(\alpha) \vee \boxed{\alpha \mathbf{R} \beta})$$

$$\text{tr}(G\alpha) = \text{tr}(\alpha) \wedge \boxed{G\alpha}$$

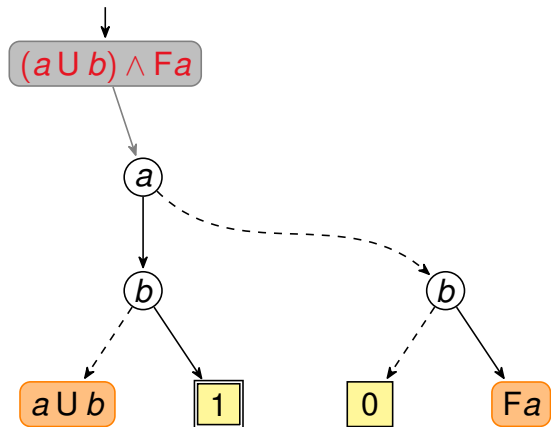
With the convention that  $\boxed{\alpha} \wedge \boxed{\beta} = \boxed{\alpha \wedge \beta}$ ,  $\boxed{\alpha} \vee \boxed{\beta} = \boxed{\alpha \vee \beta}$ , ...

# From $LTL_f$ to MTDFA: Build States One at a Time

To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .

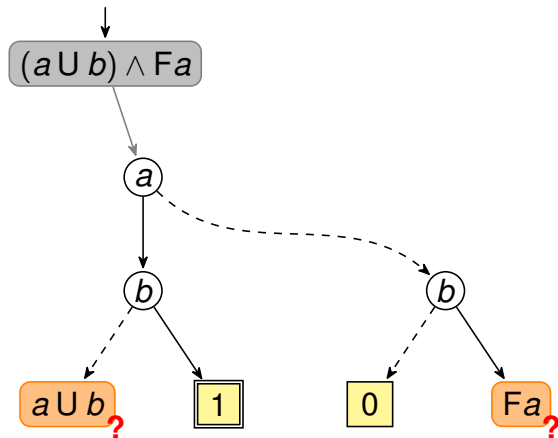
# From $LTL_f$ to MTDFA: Build States One at a Time



To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .

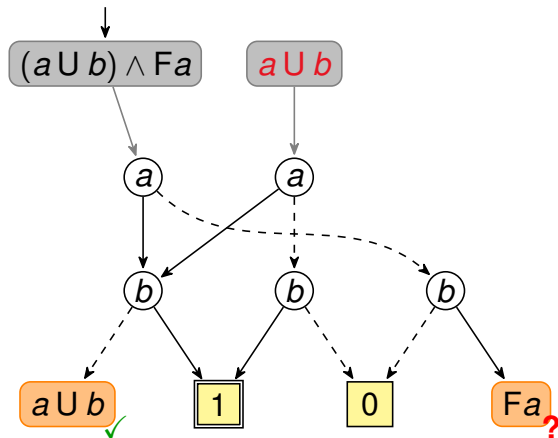
# From $LTL_f$ to MTDFA: Build States One at a Time



To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .
- 2 Compute successors for each new terminal:

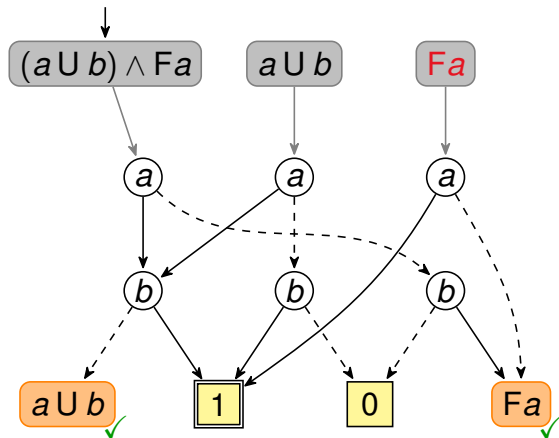
# From $LTL_f$ to MTDFA: Build States One at a Time



To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .
- 2 Compute successors for each new terminal:
  - ▶  $tr(a \cup b)$  (cached)

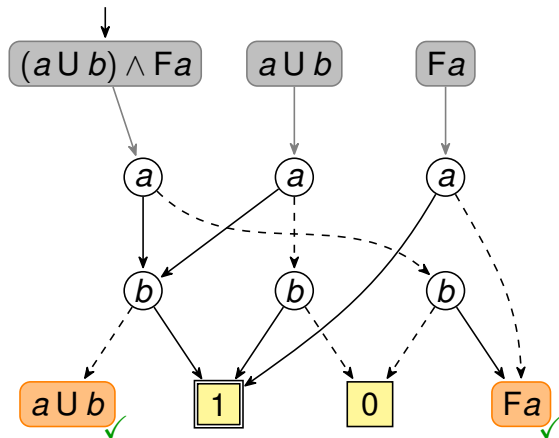
# From $LTL_f$ to MTDFA: Build States One at a Time



To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .
- 2 Compute successors for each new terminal:
  - ▶  $tr(a \cup b)$  (cached)
  - ▶  $tr(Fa)$  (cached)

# From $LTL_f$ to MTDFA: Build States One at a Time

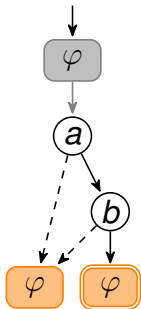


To translate  $(a \cup b) \wedge Fa$ :

- 1 Compute successors of the initial state:  $tr((a \cup b) \wedge Fa)$ .
- 2 Compute successors for each new terminal:
  - ▶  $tr(a \cup b)$  (cached)
  - ▶  $tr(Fa)$  (cached)
- 3 Done.

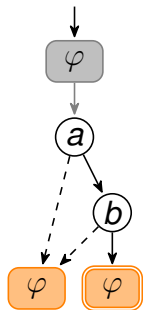
# Quick Note on Accepting Terminals

Our use of accepting terminals differs from Mona's implementation of MTDFAs.

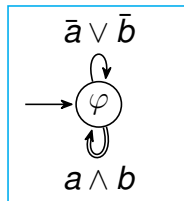


# Quick Note on Accepting Terminals

Our use of accepting terminals differs from Mona's implementation of MTDFAs.

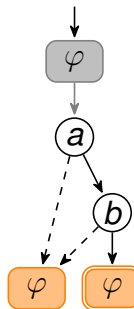


If you interpret the MTBDD roots as states, you get a transition-based DFA:

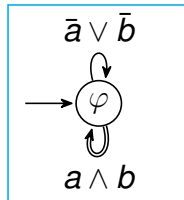


# Quick Note on Accepting Terminals

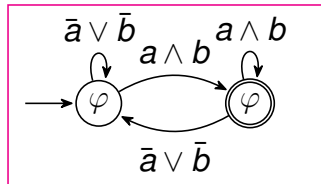
Our use of accepting terminals differs from Mona's implementation of MTDFAs.



If you interpret the MTBDD roots as states, you get a transition-based DFA:

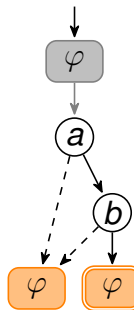


If you interpret the MTBDD terminals as states, you get a state-based DFA:

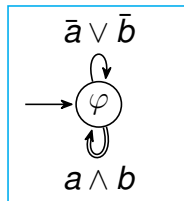


# Quick Note on Accepting Terminals

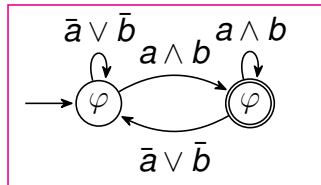
Our use of accepting terminals differs from Mona's implementation of MTDFAs.



If you interpret the MTBDD roots as states, you get a transition-based DFA:



If you interpret the MTBDD terminals as states, you get a state-based DFA:



In any case, when using an MTDFA for synthesis, accepting terminals can all be replaced by the accepting sink 1.

## What we Suggest

- ① Direct translation from  $LTL_f$  to MTBDD, building the MTDFA one state at a time. ✓
- ② Solving the game on the MTDFA directly. ✓
- ③ Doing those on-the-fly.  next slide

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathbf{G}i_1) \leftrightarrow (o_0 \wedge \mathbf{X}!\mathbf{X}!o_1)$$

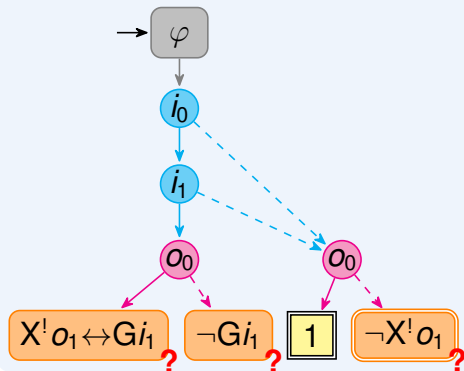
## The MTDFA

## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA

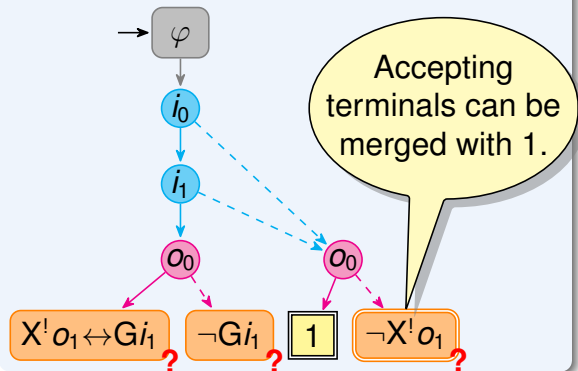


## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA

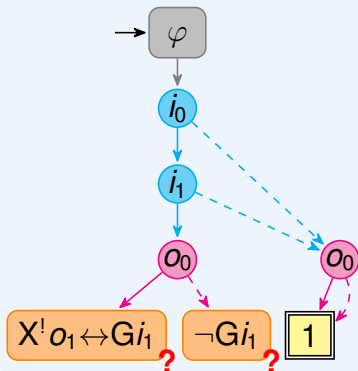


## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X!X! o_1)$$

## The MTDFA

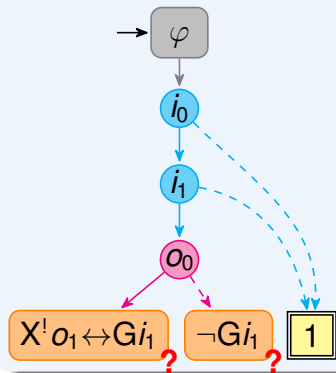


## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X!X! o_1)$$

## The MTDFA

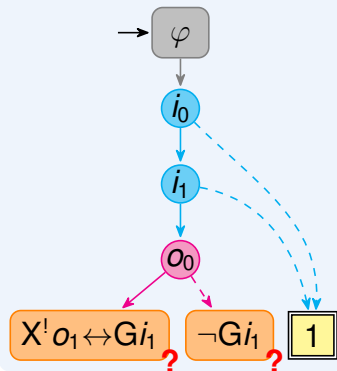


## The Game Interpretation

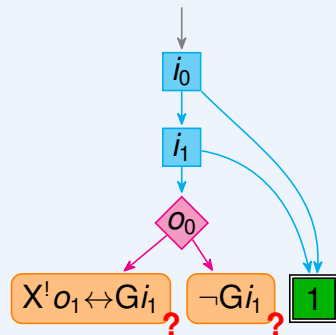
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



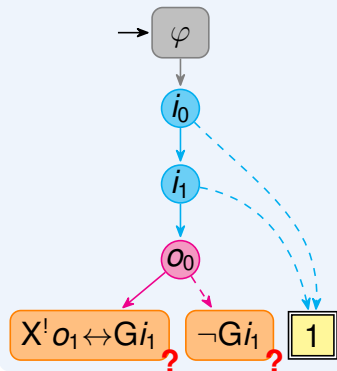
## The Game Interpretation



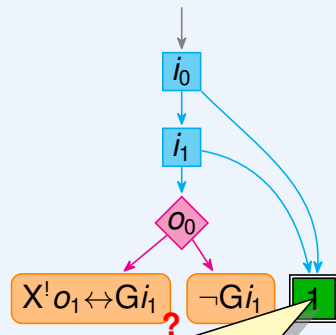
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



## The Game Interpretation

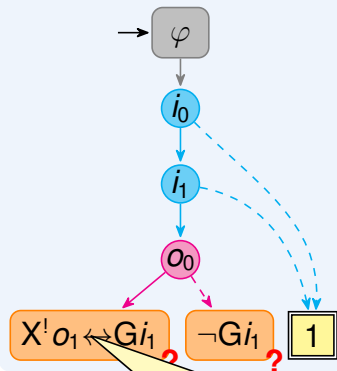


Backpropagation runs during construction.

# Building the MTDFA & Solving the Game On-The-Fly

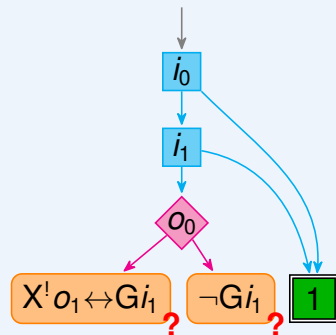
$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



Pick another terminal to develop.

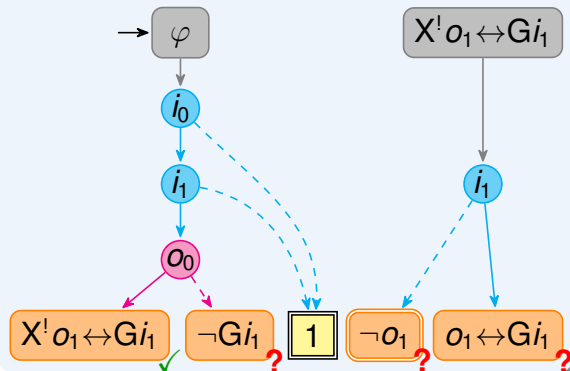
## The Game Interpretation



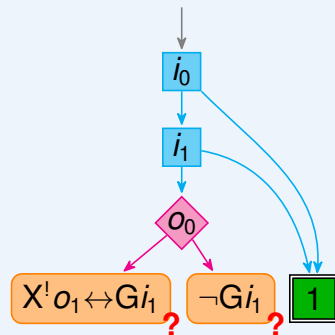
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



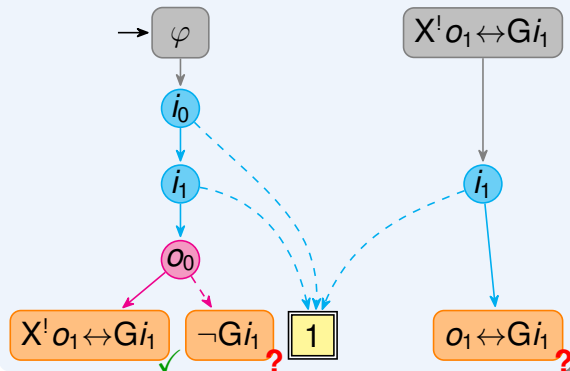
## The Game Interpretation



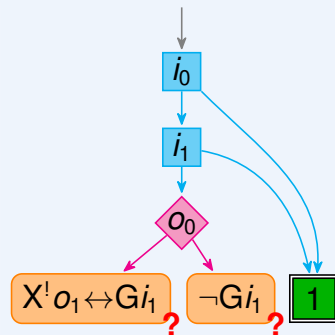
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



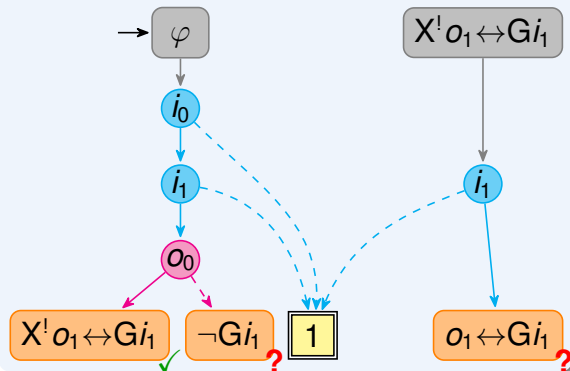
## The Game Interpretation



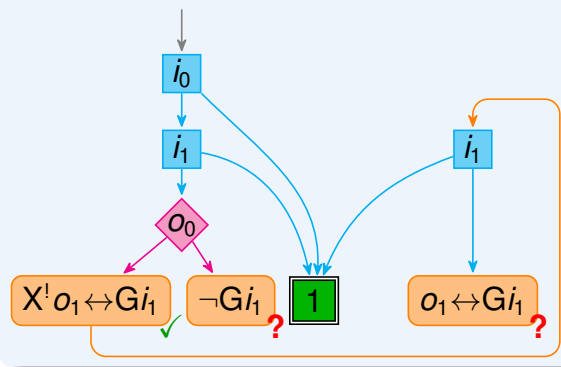
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



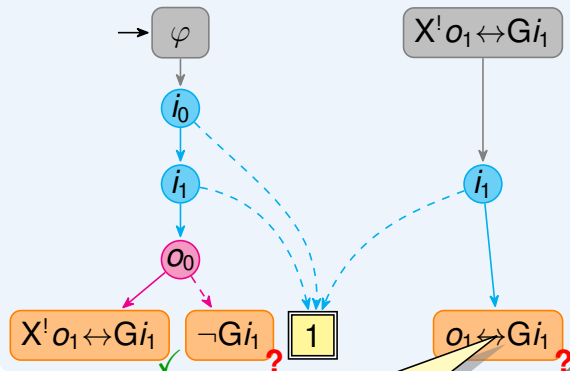
## The Game Interpretation



# Building the MTDFA & Solving the Game On-The-Fly

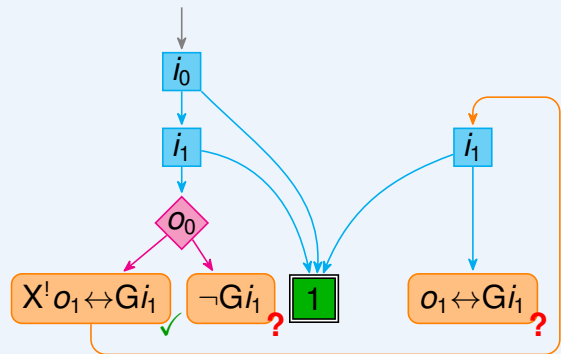
$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



Let's develop this one next.

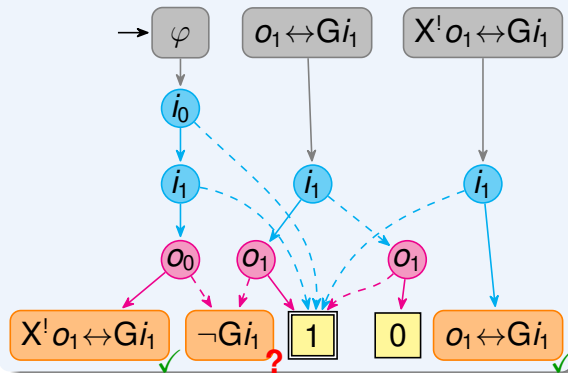
## The Game Interpretation



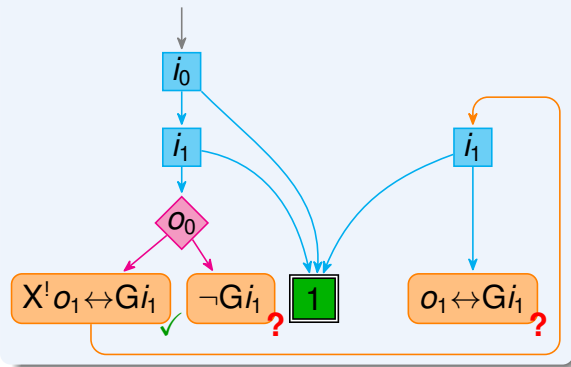
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



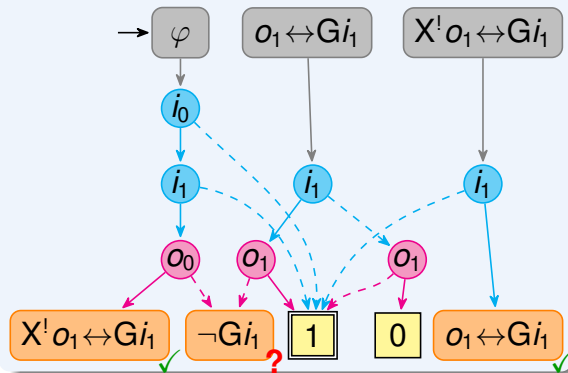
## The Game Interpretation



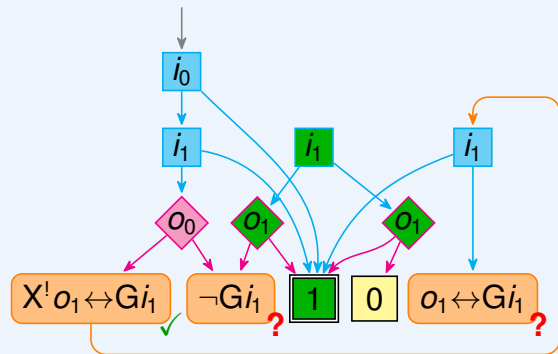
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



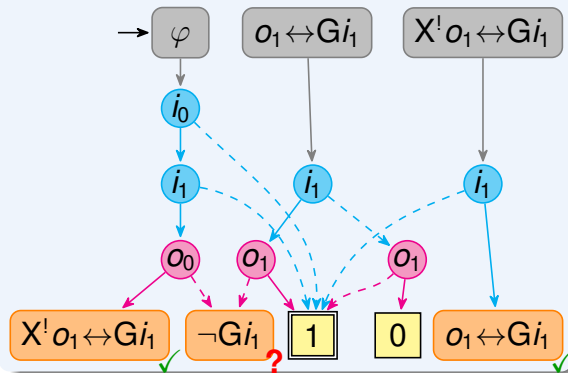
## The Game Interpretation



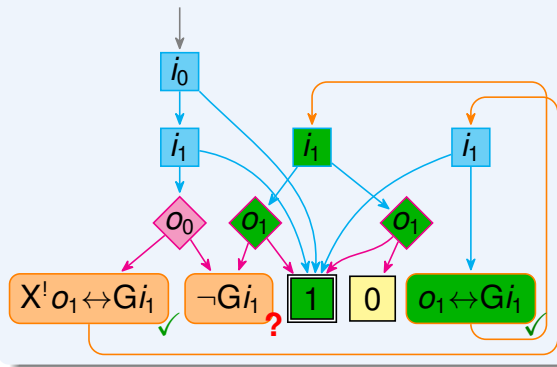
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



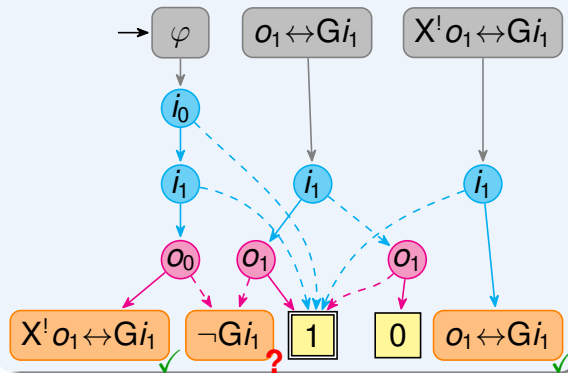
## The Game Interpretation



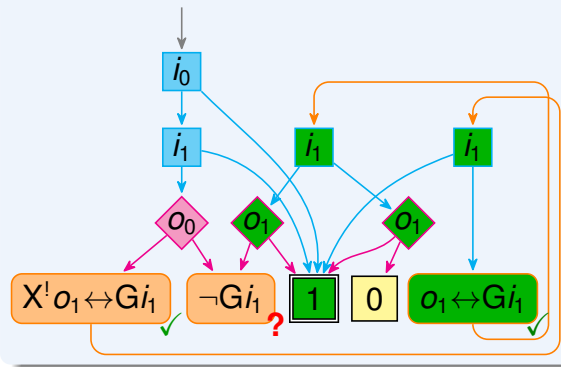
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



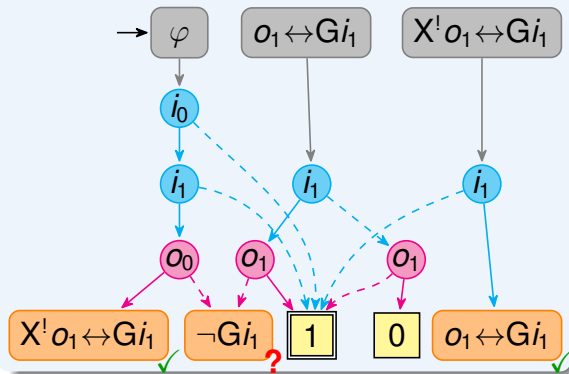
## The Game Interpretation



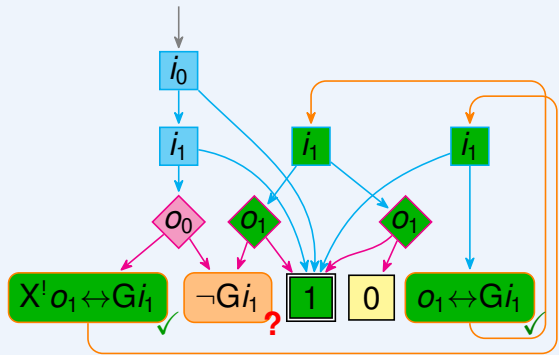
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathbf{G} i_1) \leftrightarrow (o_0 \wedge \mathbf{X}^! \mathbf{X}^! o_1)$$

# The MTDFA



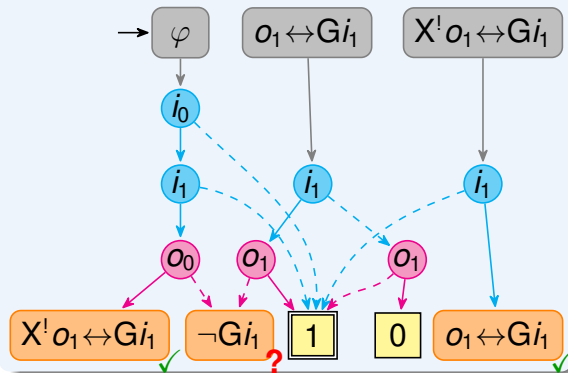
# The Game Interpretation



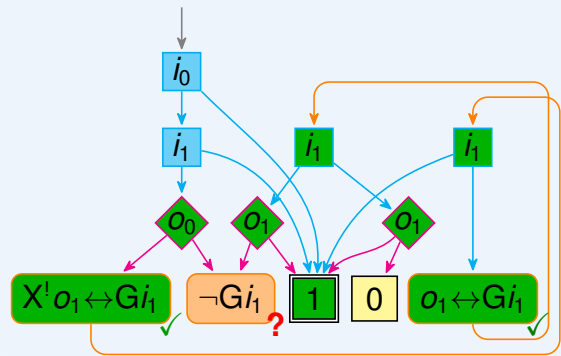
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



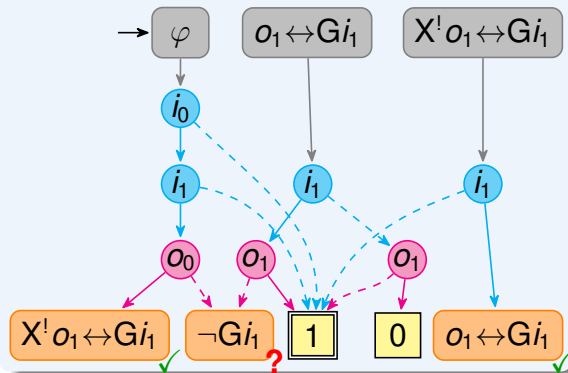
## The Game Interpretation



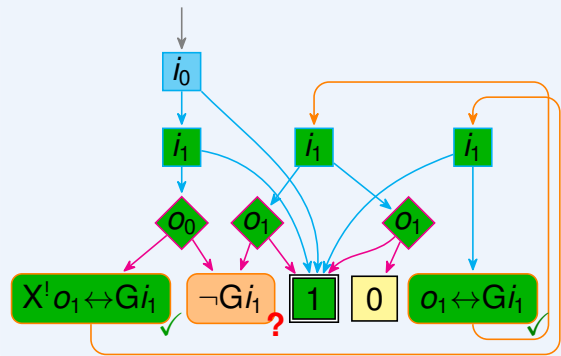
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



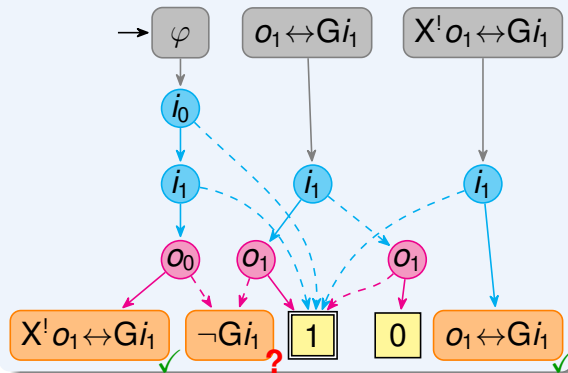
## The Game Interpretation



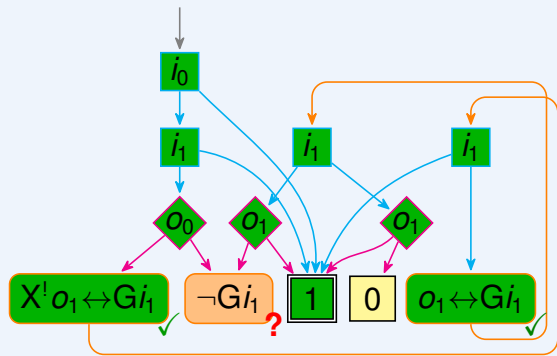
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



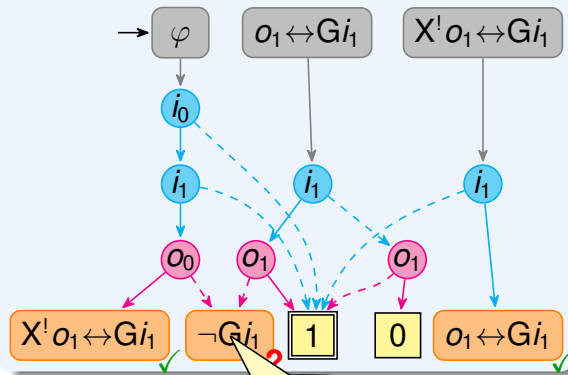
## The Game Interpretation



# Building the MTDFA & Solving the Game On-The-Fly

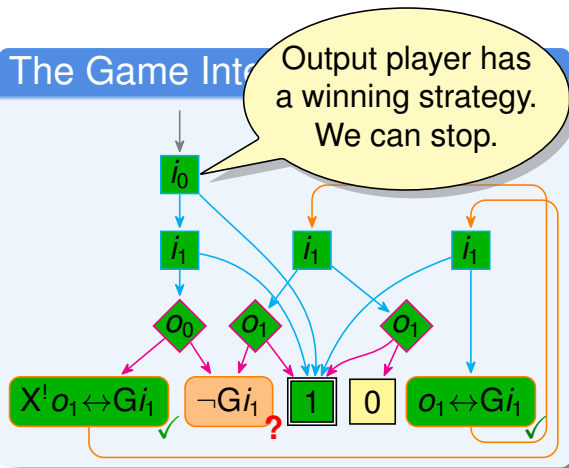
$$\varphi = (i_0 \wedge G i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$$

## The MTDFA



Never developed.

## The Game Int



# LTL<sub>f</sub> Synthesis with MTDFA

Implemented in two new tools  
distributed with Spot 2.14 [▶ www](#)

ltlfsynt

[▶ www](#)

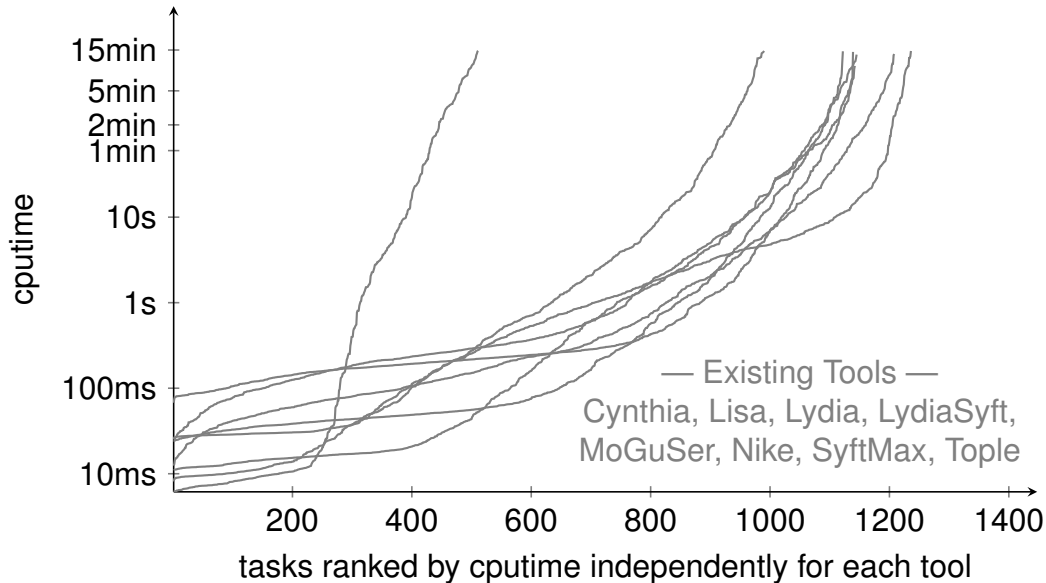
ltlf2dfa

[▶ www](#)

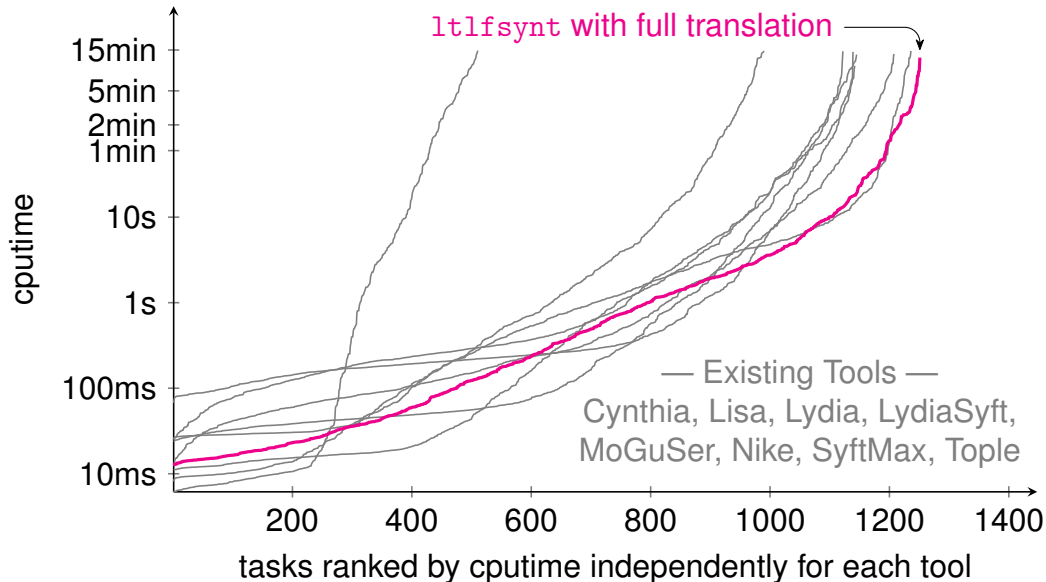
## What we Suggest

- 1 Direct translation from LTL<sub>f</sub> to MTBDD, building the MTDFA one state at a time. ✓
- 2 Solving the game on the MTDFA directly. ✓
- 3 Doing those on-the-fly. ✓

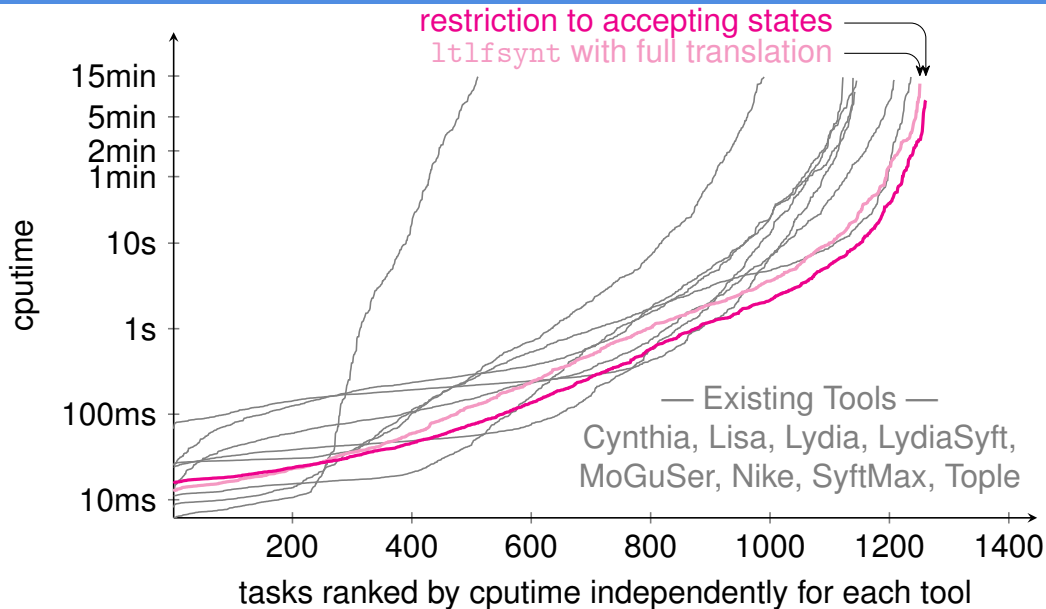
# A Benchmark (Specifications from SyntComp)



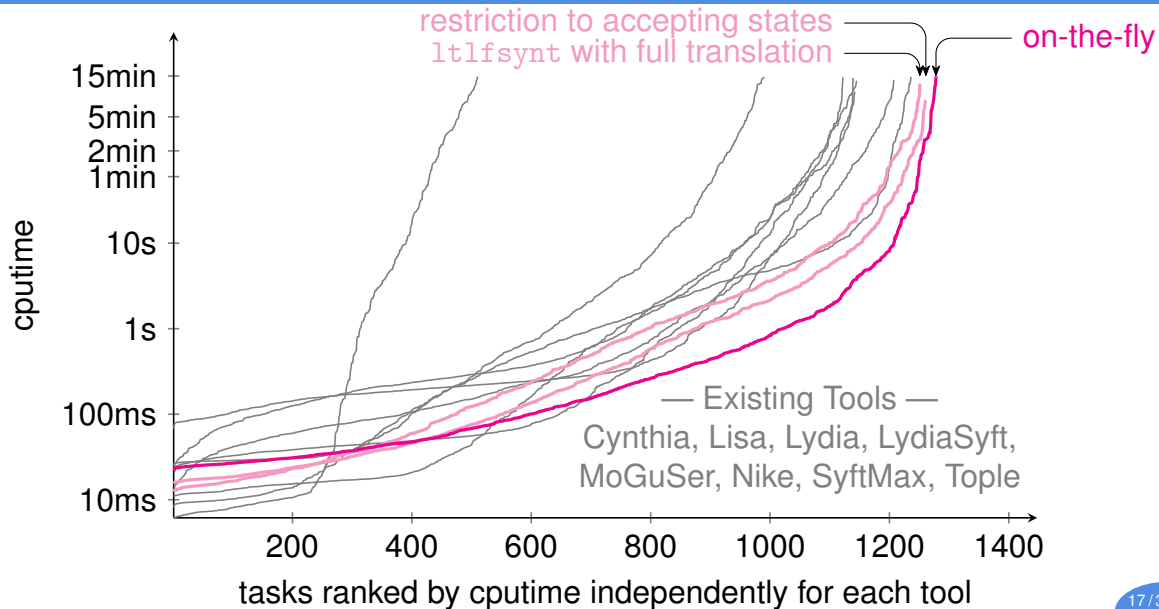
# A Benchmark (Specifications from SyntComp)



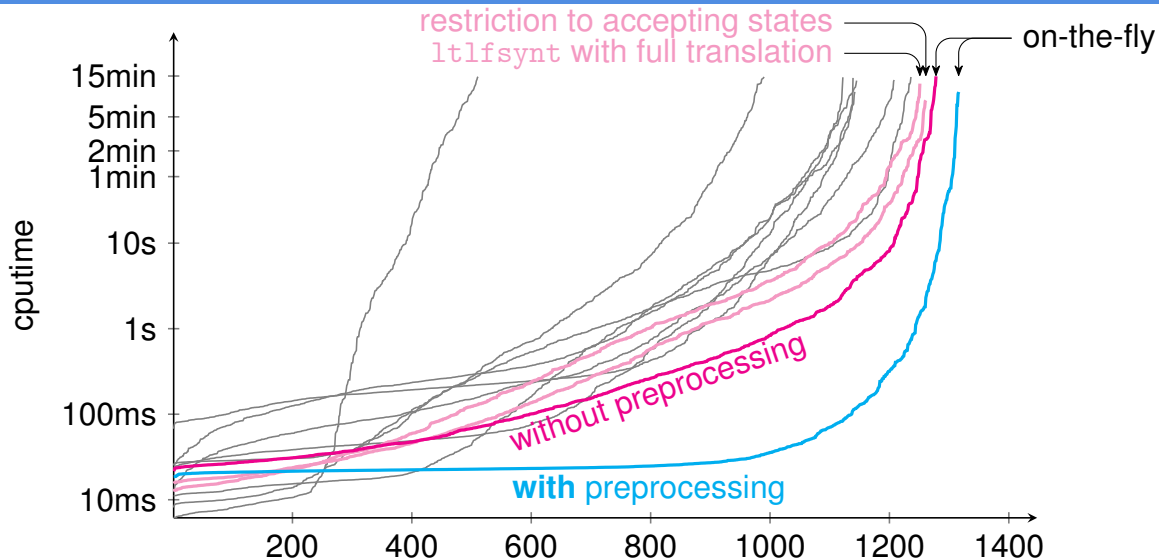
# A Benchmark (Specifications from SyntComp)



# A Benchmark (Specifications from SyntComp)



# A Benchmark (Specifications from SyntComp)



tasks ranked by cputime independently for each tool

# Conclusion (for first part)

## Efficient $LTL_f$ tools to build upon

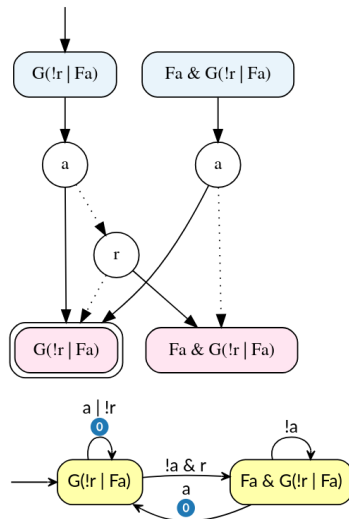
Distributed with Spot 2.14:

- ▶ `ltlf2dfa`
- ▶ `ltlfsynt` (🏆 won SyntComp'25 and '26)
- ▶ C++ & Python APIs available

## Ideas to take away

- ▶ MTBDDs are great for deterministic automata with propositional alphabets.
- ▶ Such automata can be interpreted as games at the level of MTBDD nodes (deciding one proposition at a time).

```
In [20]: a3 = spot.ltlf_to_mtdfa("G(!r | Fa)")  
display(a3, a3.as_twa())
```



# Outline

## 1 LTL<sub>f</sub> Synthesis

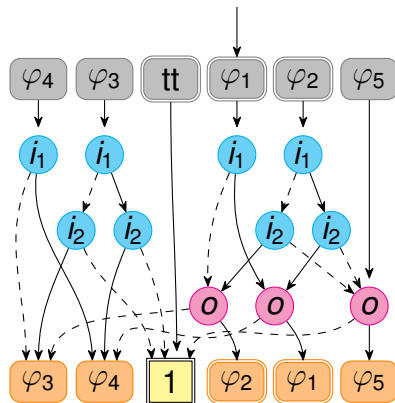
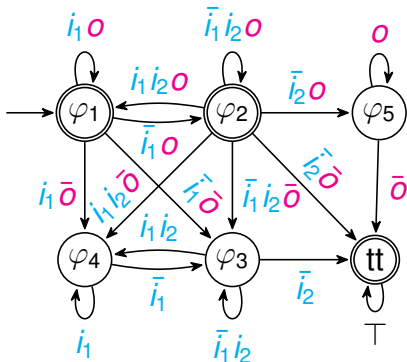
## 2 Obligation Synthesis (LTL)

- MTBDD-based DBA with State-Based Acceptance
- Temporal Hierarchy
- Contributions
- The Obligation class
- From Obligation to MTDBA
- Solving Weak Games
- Speedup

## 3 Synthesis Under Partial Observability



# MTBDD-based DBA with State-Based Acceptance



$$\varphi_1 = G(i_1 \vee X i_2) \leftrightarrow G o$$

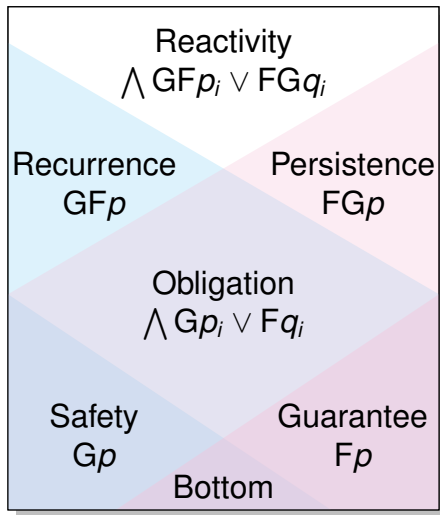
$$\varphi_2 = (i_2 \wedge G(i_1 \vee X i_2)) \leftrightarrow G o$$

$$\varphi_3 = \neg(i_2 \wedge G(i_1 \vee X i_2))$$

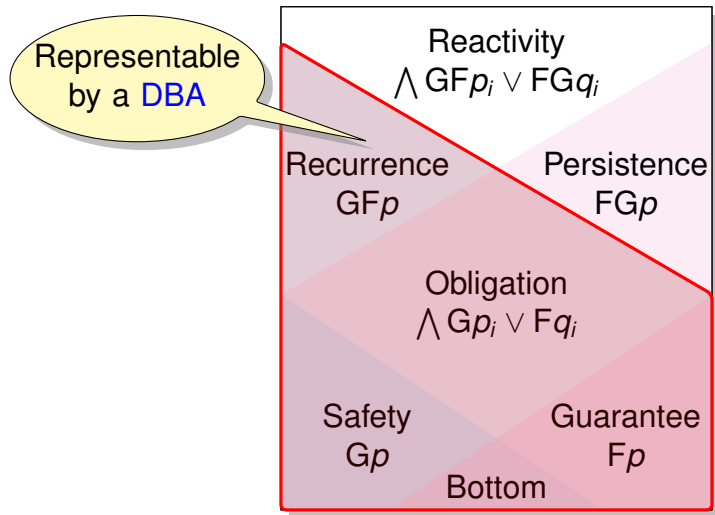
$$\varphi_4 = \neg G(i_1 \vee X i_2)$$

$$\varphi_5 = \neg G o$$

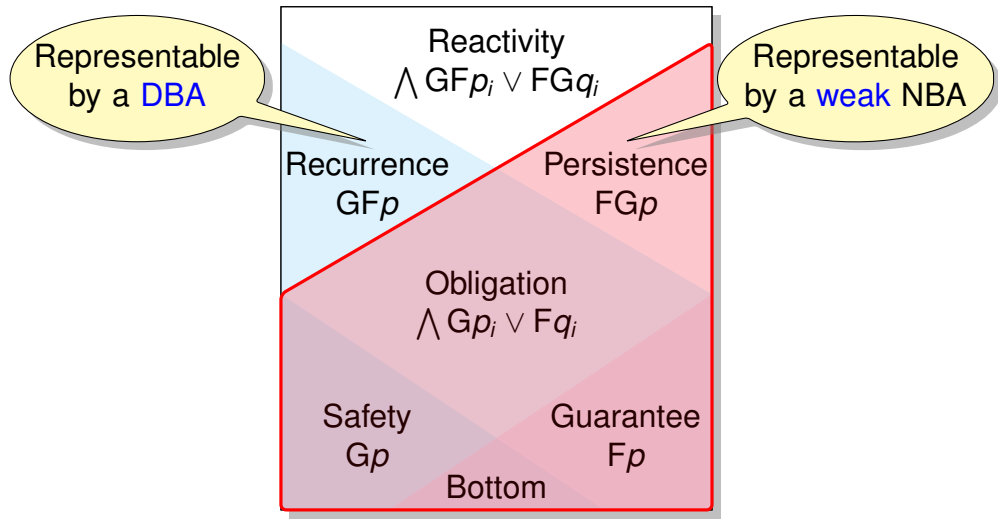
# Temporal Hierarchy for LTL and $\omega$ -Automata



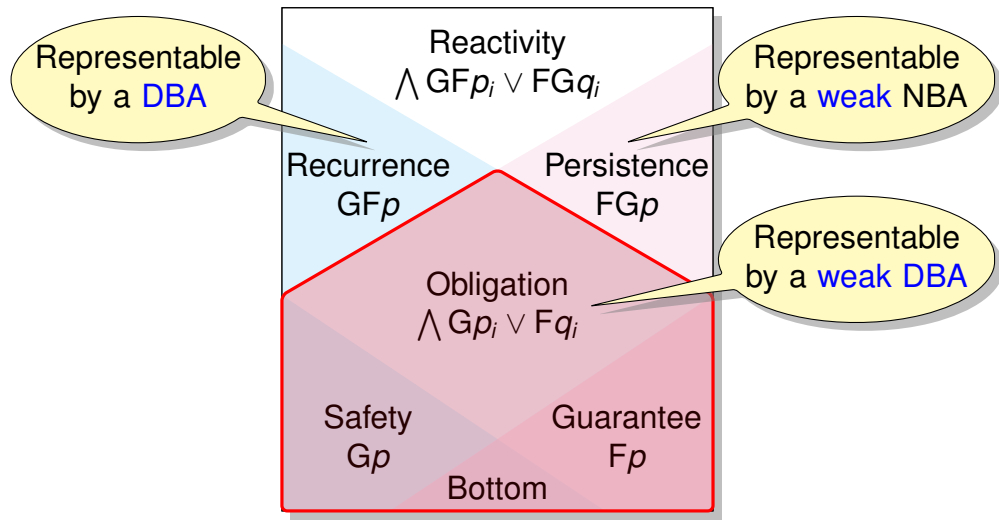
# Temporal Hierarchy for LTL and $\omega$ -Automata



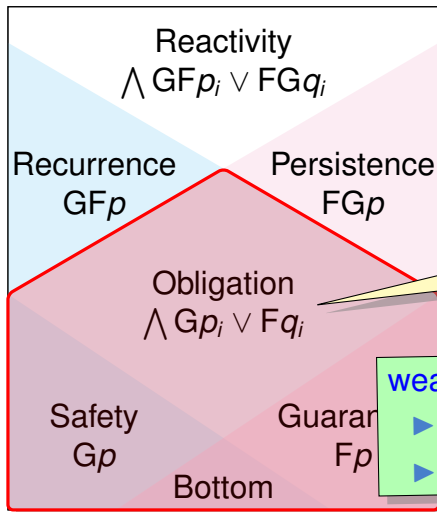
# Temporal Hierarchy for LTL and $\omega$ -Automata



# Temporal Hierarchy for LTL and $\omega$ -Automata



# Temporal Hierarchy for LTL and $\omega$ -Automata

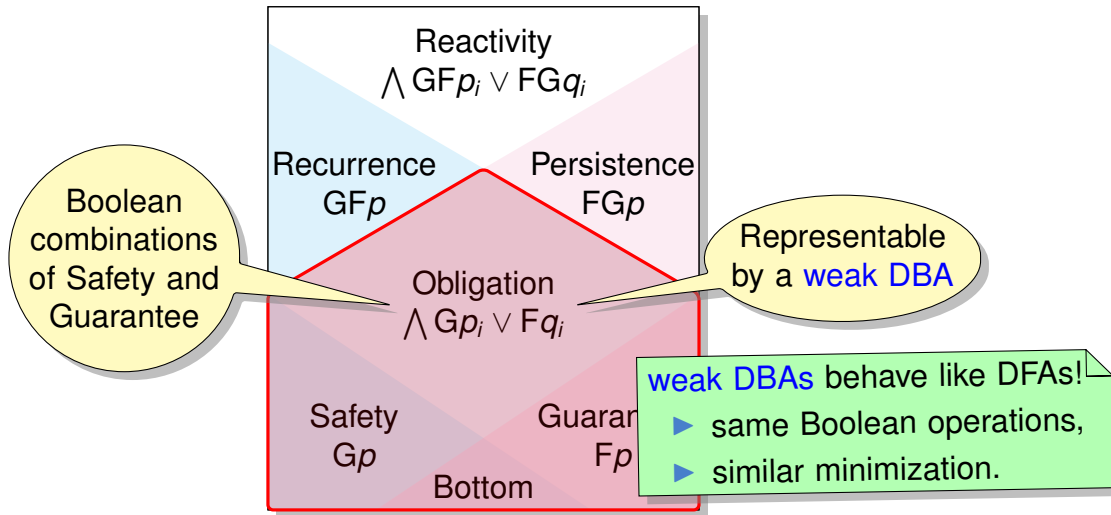


Representable  
by a **weak DBA**

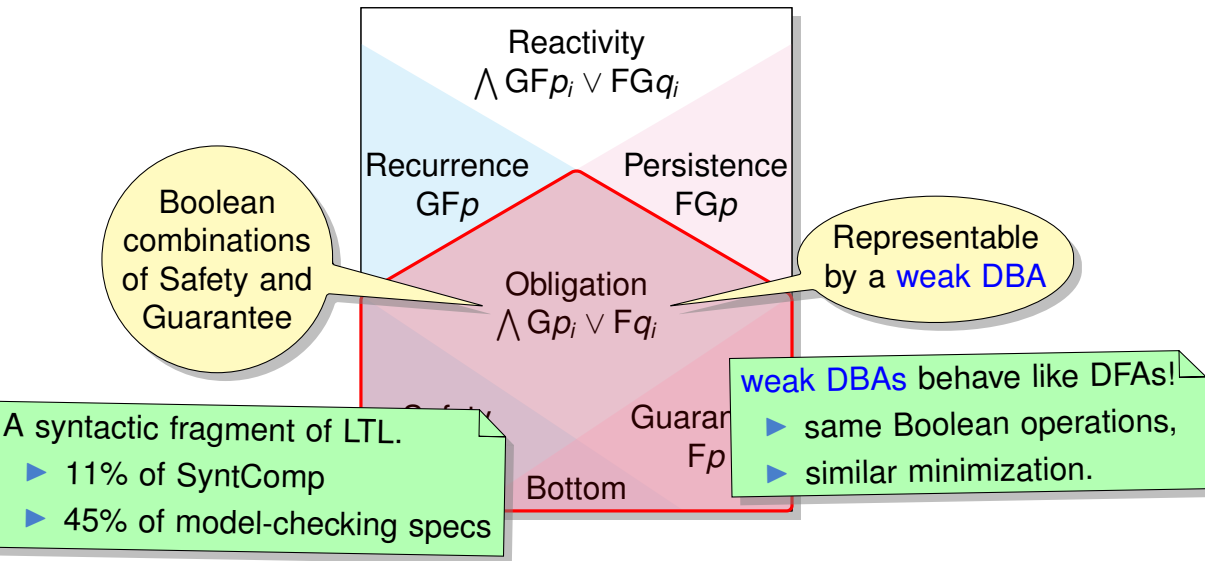
**weak DBAs** behave like DFAs!

- ▶ same Boolean operations,
- ▶ similar minimization.

# Temporal Hierarchy for LTL and $\omega$ -Automata



# Temporal Hierarchy for LTL and $\omega$ -Automata



# Contributions

## A new translation from syntactic obligations to weak DBA

- ▶ Produces an **MTBDD-encoded weak DBA** directly.
- ▶ States are labeled by obligation formulas.
- ▶ Acceptance of a state is decided by looking at the Boolean combination of safety and guarantee properties its formula represents.
- ▶ **Minimization is cheap** in practice.

## Faster reactive synthesis for syntactic obligations

- ▶ Translate the specification to MTBDD **on-the-fly**...
- ▶ ... while solving a **weak game** (still linear).

All of this is implemented in Spot 2.15



▶ [www](#)

# Obligations = Weak Deterministic Büchi Automata

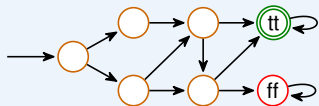
Bottom (Safety  $\cap$  Guarantee)

$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B$   
 $\mid X\varphi_B$

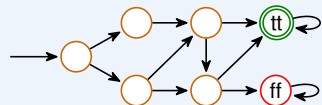


DBA is a DAG,  
until  $\textcircled{\text{tt}}$  or  $\textcircled{\text{ff}}$

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B$   
 $\mid X\varphi_B$



DBA is a DAG,  
until  $\textcircled{\text{tt}}$  or  $\textcircled{\text{ff}}$

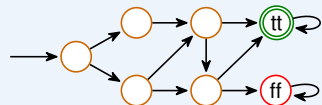
## Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S$   
 $\mid X\varphi_S \mid \mathbf{G}\varphi_S \mid \varphi_S \mathbf{R} \varphi_S$

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

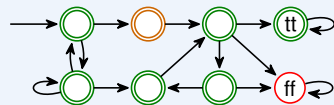
$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B$   
 $\mid X\varphi_B$



DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S$   
 $\mid X\varphi_S \mid G\varphi_S \mid \varphi_S R \varphi_S$

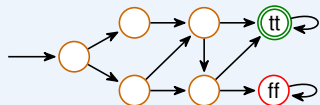


all states but  $\text{ff}$   
are accepting

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg \varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



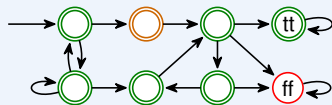
DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Guarantee

$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$

## Safety

$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$

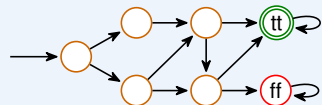


all states but  $\text{ff}$   
are accepting

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

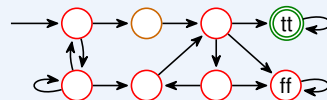
$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg \varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Guarantee

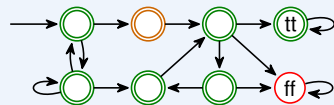
$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$



all states but  $\text{tt}$   
are rejecting

## Safety

$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$

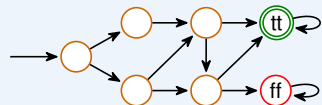


all states but  $\text{ff}$   
are accepting

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

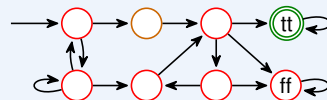
$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Guarantee

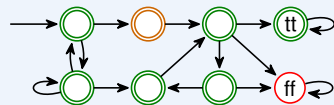
$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$



all states but  $\text{tt}$   
are rejecting

## Safety

$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$

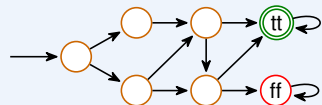


all states but  $\text{ff}$   
are accepting

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

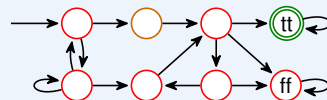
$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Guarantee

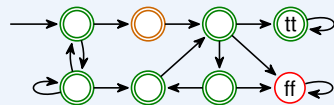
$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$



all states but  $\text{tt}$   
are rejecting

## Safety

$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$



all states but  $\text{ff}$   
are accepting

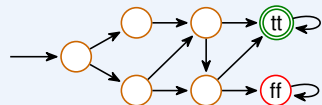
## Obligation

$$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \\ \mid X\varphi_O$$

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

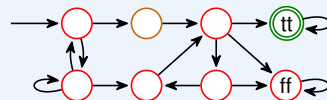
$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



DBA is a DAG,  
until  $\textcircled{\text{tt}}$  or  $\textcircled{\text{ff}}$

## Guarantee

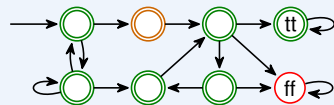
$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$



all states but  $\textcircled{\text{tt}}$   
are rejecting

## Safety

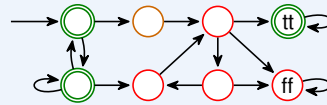
$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$



all states but  $\textcircled{\text{ff}}$   
are accepting

## Obligation

$$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \\ \mid X\varphi_O$$

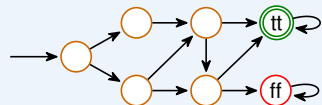


each SCC  
contains only  
accepting or  
rejecting states

# Obligations = Weak Deterministic Büchi Automata

## Bottom (Safety $\cap$ Guarantee)

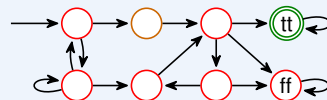
$$\varphi_B ::= \text{ff} \mid \text{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \\ \mid X\varphi_B$$



DBA is a DAG,  
until  $\text{tt}$  or  $\text{ff}$

## Guarantee

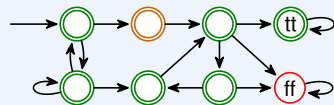
$$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \\ \mid X\varphi_G \mid F\varphi_G \mid \varphi_G \text{ U } \varphi_G$$



all states but  $\text{tt}$   
are rejecting

## Safety

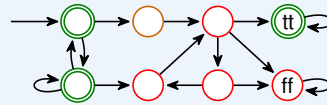
$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \\ \mid X\varphi_S \mid G\varphi_S \mid \varphi_S \text{ R } \varphi_S$$



all states but  $\text{ff}$   
are accepting

## Obligation

$$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \\ \mid X\varphi_O \mid \varphi_O \text{ U } \varphi_G \mid \varphi_O \text{ R } \varphi_S$$



each SCC  
contains only  
accepting or  
rejecting states

# Deciding acceptance of an obligation-labeled state

$$\lambda(\text{ff}) = \perp$$

$$\lambda(\text{tt}) = \top$$

$$\lambda(\mathbf{G}\alpha) = \top$$

$$\lambda(\alpha \mathbf{R} \beta) = \top$$

$$\lambda(\mathbf{F}\alpha) = \perp$$

$$\lambda(\alpha \mathbf{U} \beta) = \perp$$

$$\lambda(\neg\alpha) = \neg\lambda(\alpha)$$

$$\lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta)$$

$$\lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

# Deciding acceptance of an obligation-labeled state

$$\lambda(\text{ff}) = \perp$$

$$\lambda(\text{tt}) = \top$$

$$\lambda(p) = *$$

$$\lambda(X\alpha) = *$$

$$\lambda(\neg\alpha) = \neg\lambda(\alpha)$$

$$\lambda(G\alpha) = \top$$

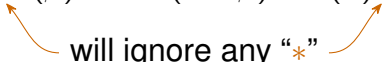
$$\lambda(\alpha \mathbf{R} \beta) = \top$$

$$\lambda(F\alpha) = \perp$$

$$\lambda(\alpha \mathbf{U} \beta) = \perp$$

$$\lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta) \quad \lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

will ignore any “\*”



# Deciding acceptance of an obligation-labeled state

$$\lambda(\text{ff}) = \perp$$

$$\lambda(\mathbf{G}\alpha) = \top$$

$$\lambda(\mathbf{F}\alpha) = \perp$$

$$\lambda(\text{tt}) = \top$$

$$\lambda(\alpha \mathbf{R} \beta) = \top$$

$$\lambda(\alpha \mathbf{U} \beta) = \perp$$

$$\lambda(p) = *$$

$$\lambda(\mathbf{X}\alpha) = *$$

$$\lambda(\neg\alpha) = \neg\lambda(\alpha)$$

$$\lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta)$$

$$\lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

will ignore any “\*”

Example:

$$\lambda(\varphi_2) = \lambda((\underbrace{i_2}_{*} \wedge \underbrace{\mathbf{G}(i_1 \vee \mathbf{X}i_2)}_{\top})) \leftrightarrow \underbrace{\mathbf{Go}}_{\top} = \top$$

follows from the above rules

# MTBDD translation is straightforward

$$\text{tr}(\text{ff}) = \boxed{0}$$

$$\text{tr}(\text{tt}) = \boxed{1}$$

$$\text{tr}(p) = \begin{array}{c} \textcircled{p} \\ \swarrow \quad \searrow \\ \boxed{0} \quad \boxed{1} \end{array} \quad \text{for } p \in \mathcal{I} \cup \mathcal{O}$$

$$\text{tr}(\textcolor{blue}{X}\alpha) = \boxed{\alpha}$$

$$\text{tr}(\textcolor{blue}{\neg}\alpha) = \textcolor{red}{\neg}\text{tr}(\alpha)$$

LTL operator

MTBDD operator

$$\text{tr}(\alpha \odot \beta) = \text{tr}(\alpha) \odot \text{tr}(\beta) \text{ for any } \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$$

$$\text{tr}(\alpha \textcolor{blue}{U} \beta) = \text{tr}(\beta) \vee (\text{tr}(\alpha) \wedge \boxed{\alpha \textcolor{blue}{U} \beta})$$

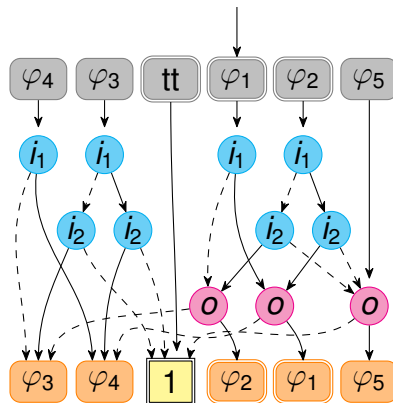
$$\text{tr}(\text{F}\alpha) = \text{tr}(\alpha) \vee \boxed{\text{F}\alpha}$$

$$\text{tr}(\alpha \textcolor{blue}{R} \beta) = \text{tr}(\beta) \wedge (\text{tr}(\alpha) \vee \boxed{\alpha \textcolor{blue}{R} \beta})$$

$$\text{tr}(\text{G}\alpha) = \text{tr}(\alpha) \wedge \boxed{\text{G}\alpha}$$

Acceptance of a state labeled by  $\alpha$  is  $\lambda(\alpha)$ .

# MTBDD-based DBA with State-Based Acceptance



$$\varphi_1 = G(i_1 \vee X i_2) \leftrightarrow G o$$

$$\varphi_2 = (i_2 \wedge G(i_1 \vee X i_2)) \leftrightarrow G o$$

$$\varphi_3 = \neg(i_2 \wedge G(i_1 \vee X i_2))$$

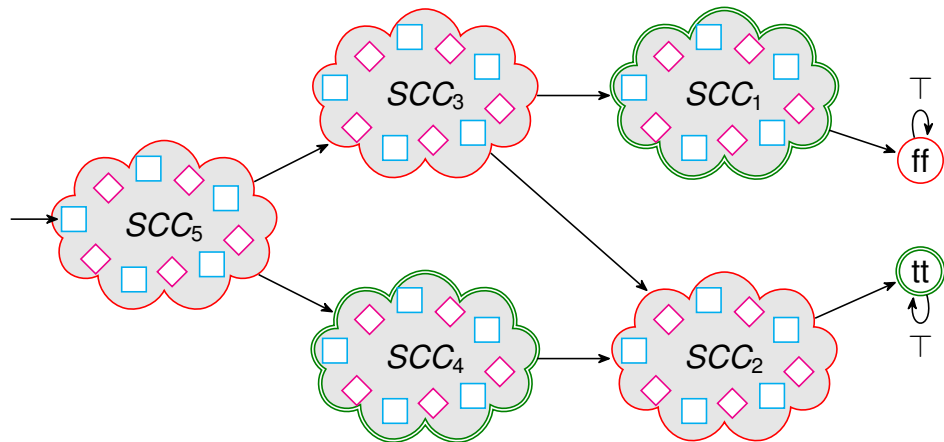
$$\varphi_4 = \neg G(i_1 \vee X i_2)$$

$$\varphi_5 = \neg G o$$

# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

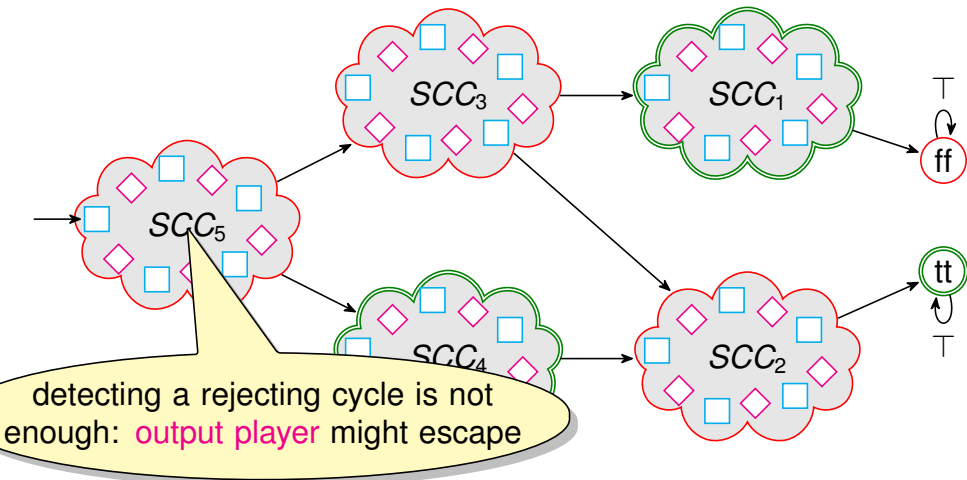
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

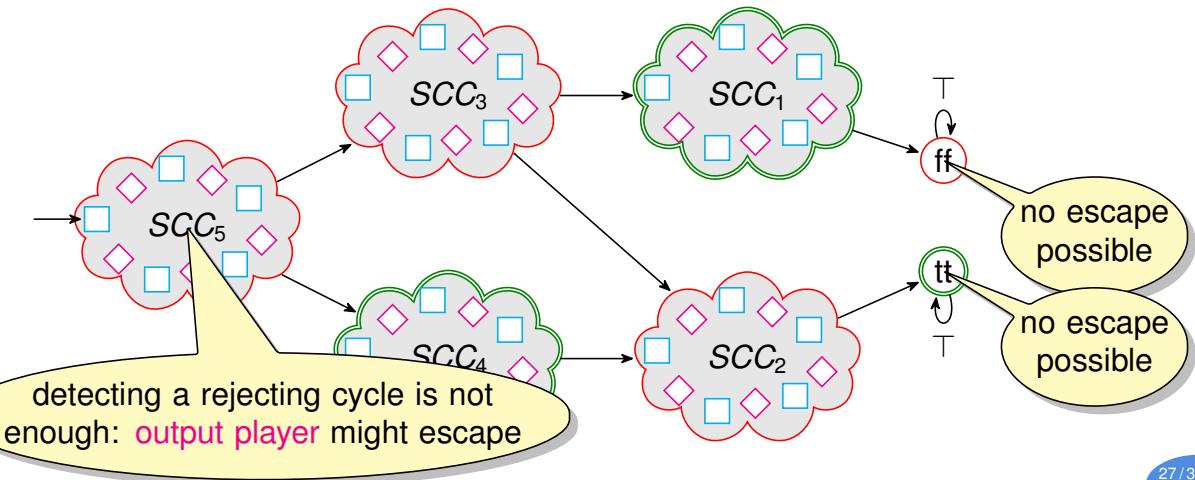
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

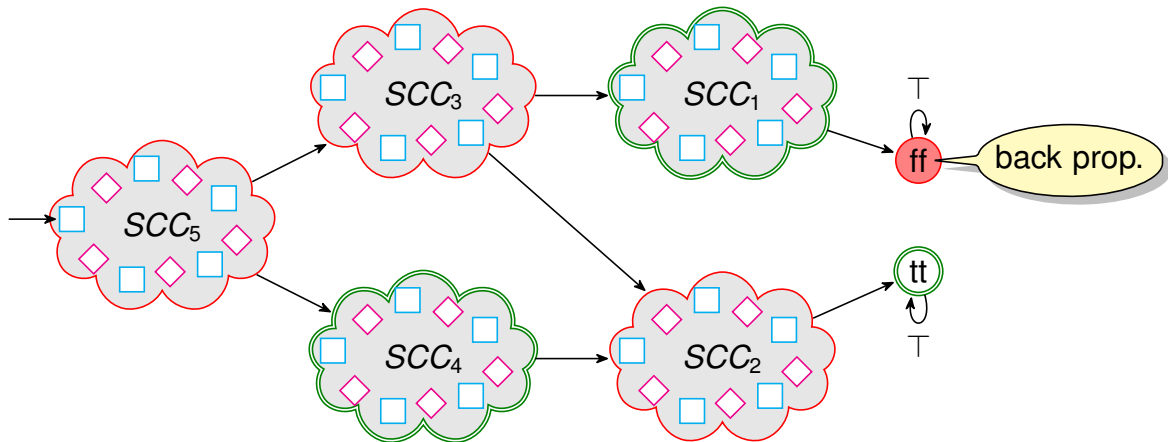
The **input player** wants the play to get stuck in a **rejecting SCC**.

The **output player** wants the play to get stuck in an **accepting SCC**.



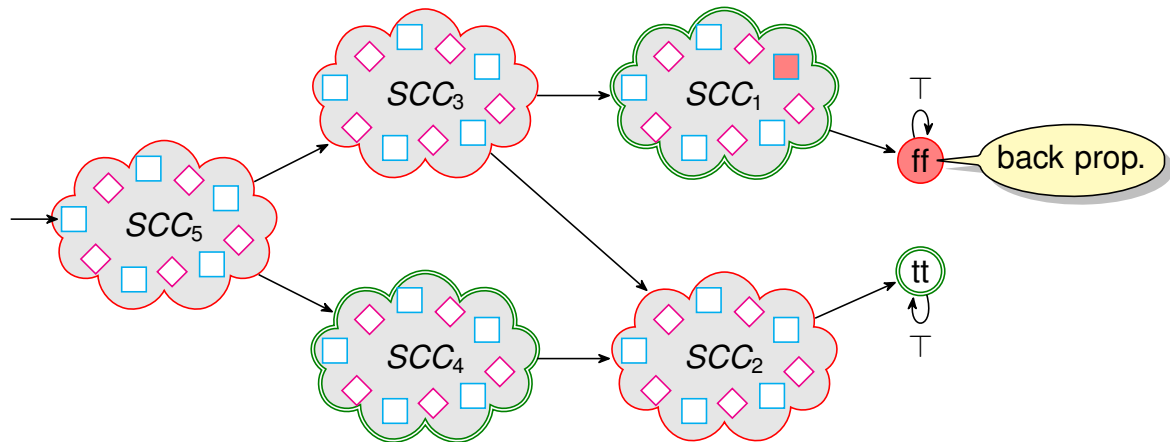
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



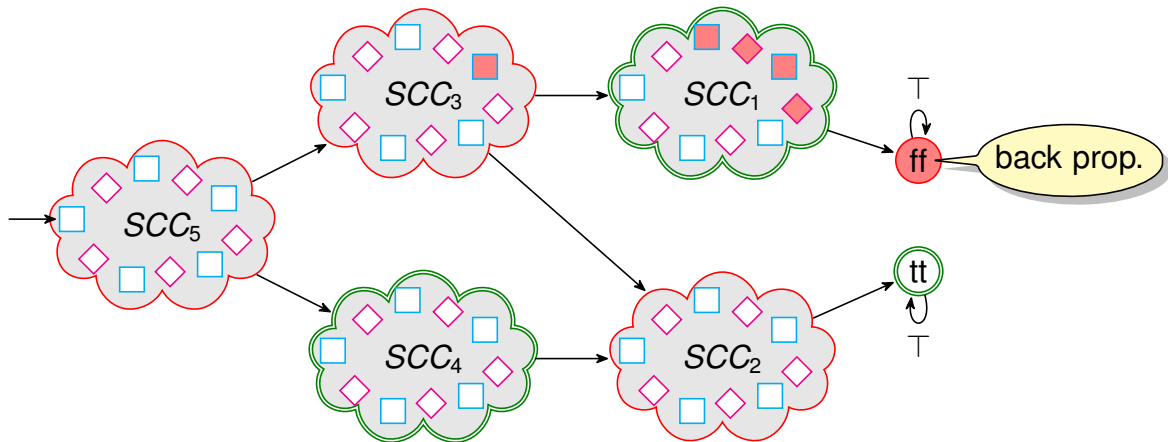
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

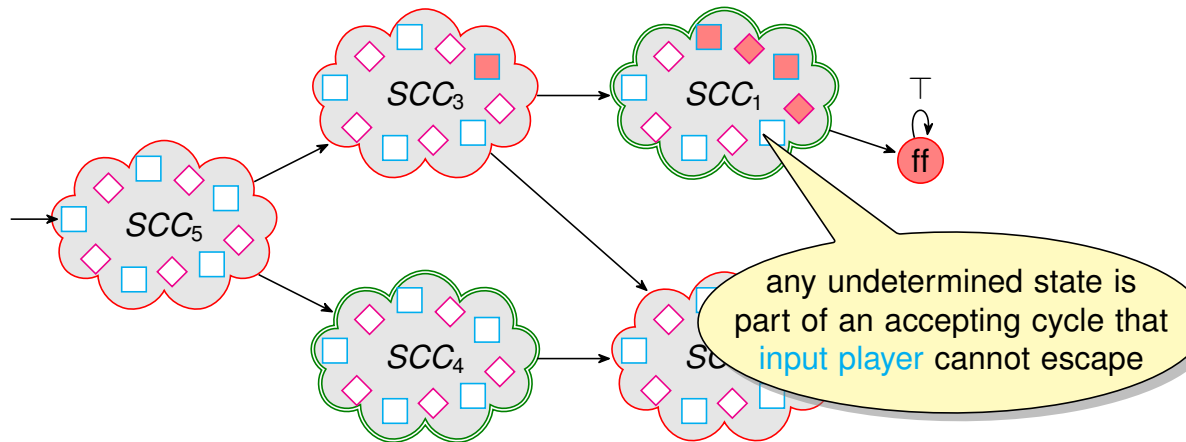
The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

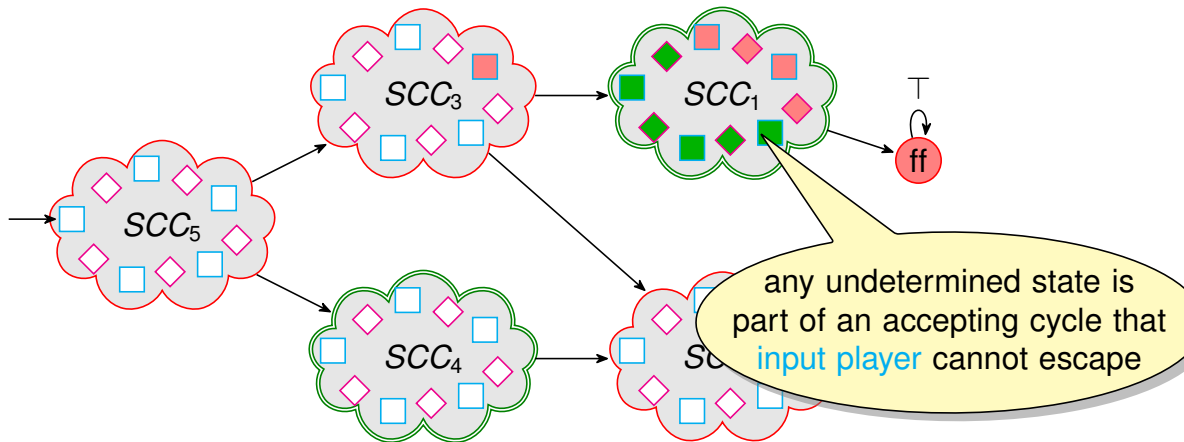
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

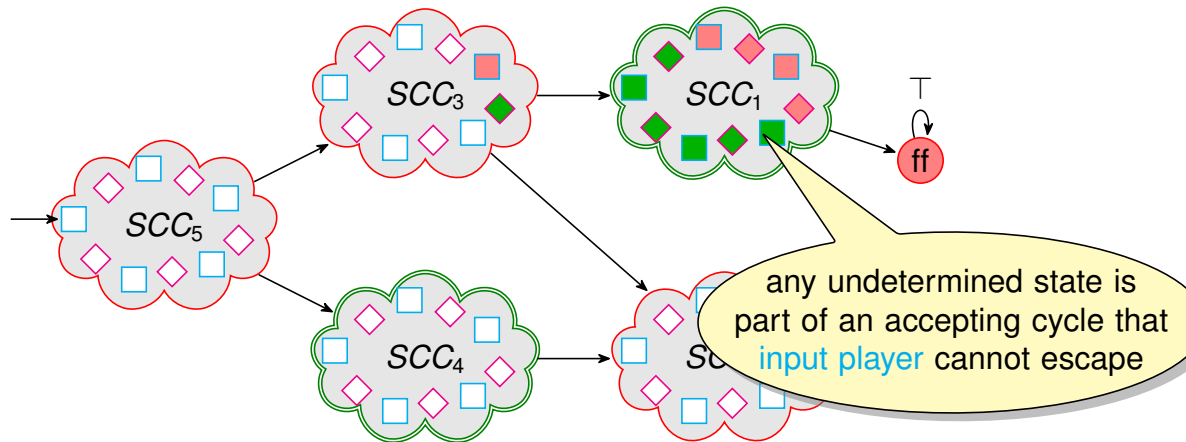
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

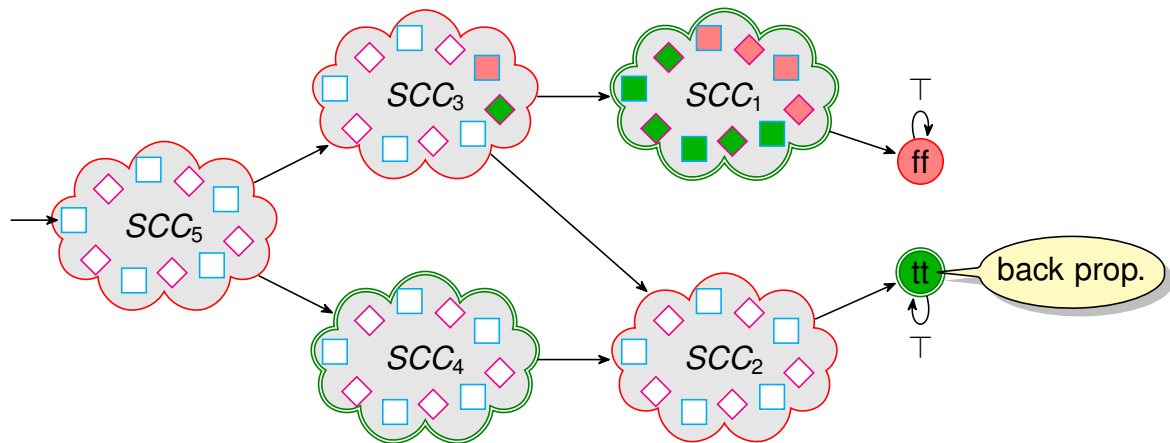
The **input player** wants the play to get stuck in a **rejecting SCC**.

The **output player** wants the play to get stuck in an **accepting SCC**.



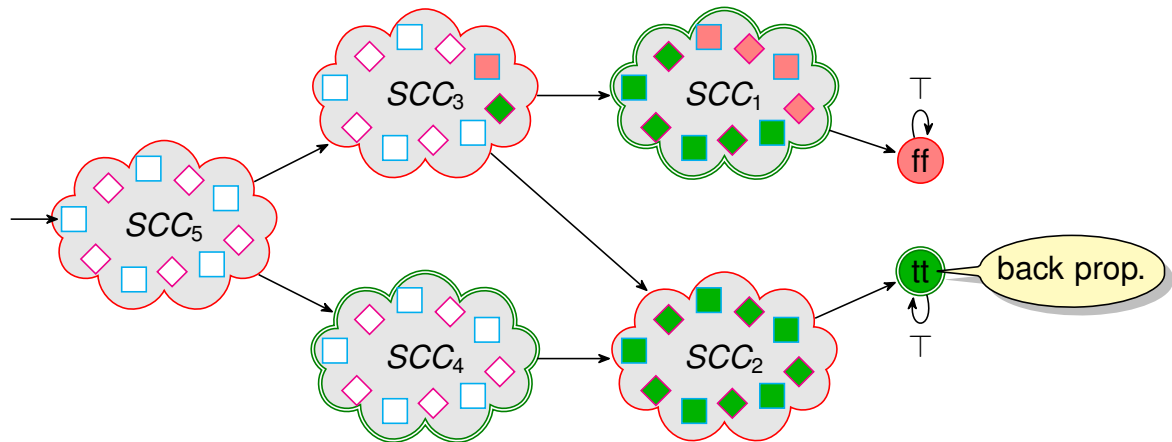
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



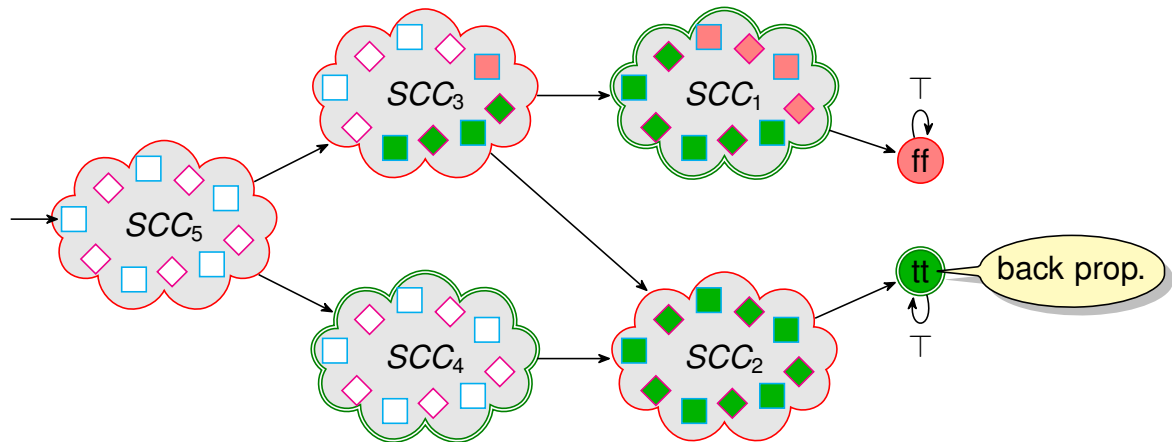
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

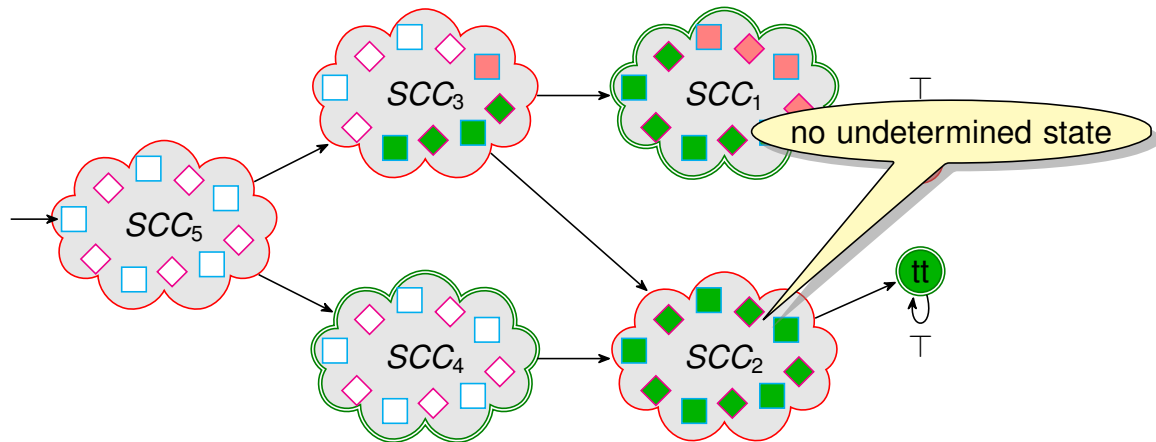
The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

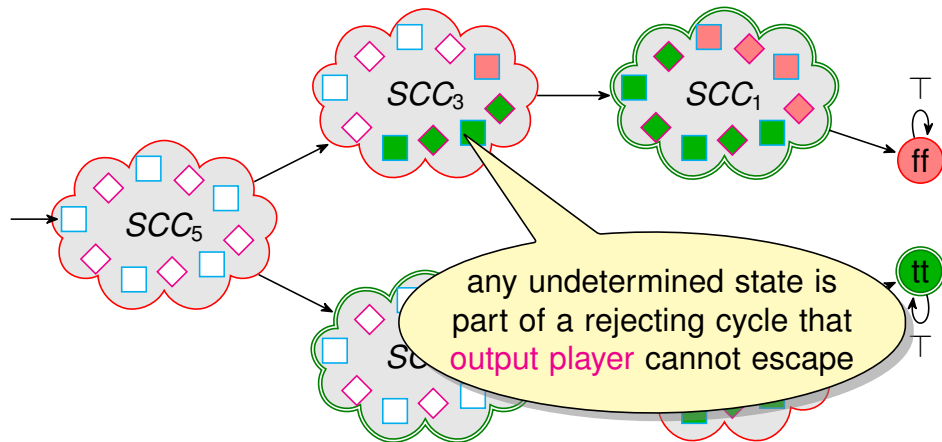
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

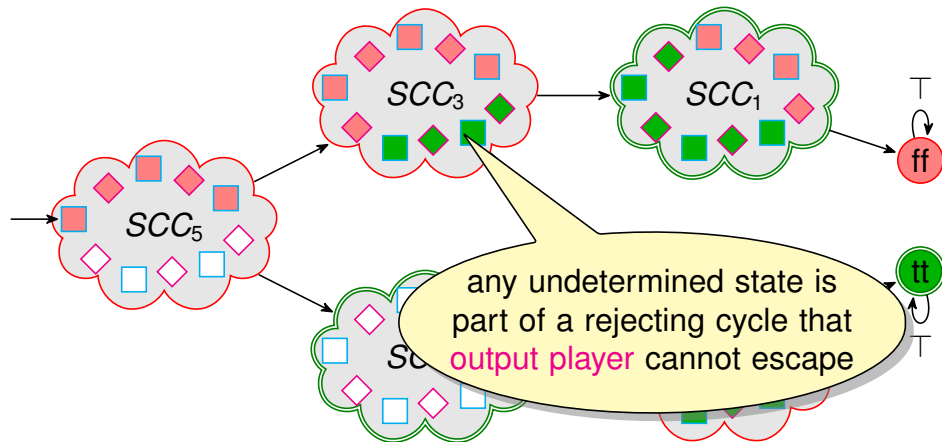
The **input player** wants the play to get stuck in a **rejecting SCC**.

The **output player** wants the play to get stuck in an **accepting SCC**.



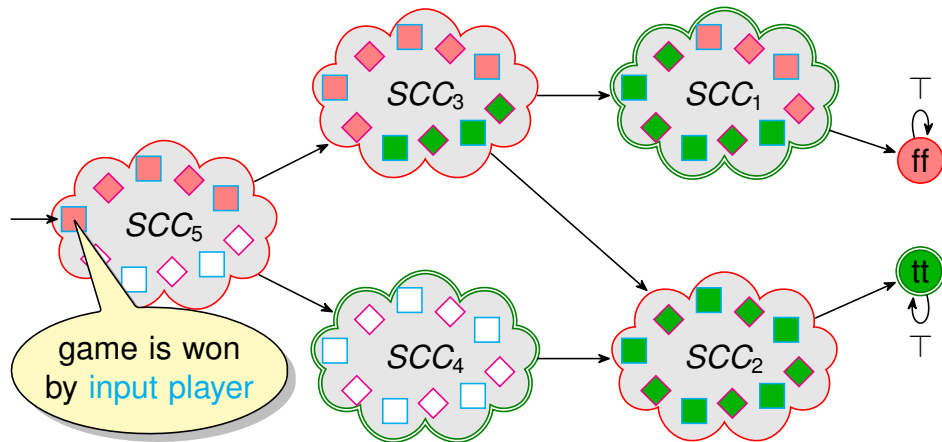
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



# Solving Weak Games Bottom-Up

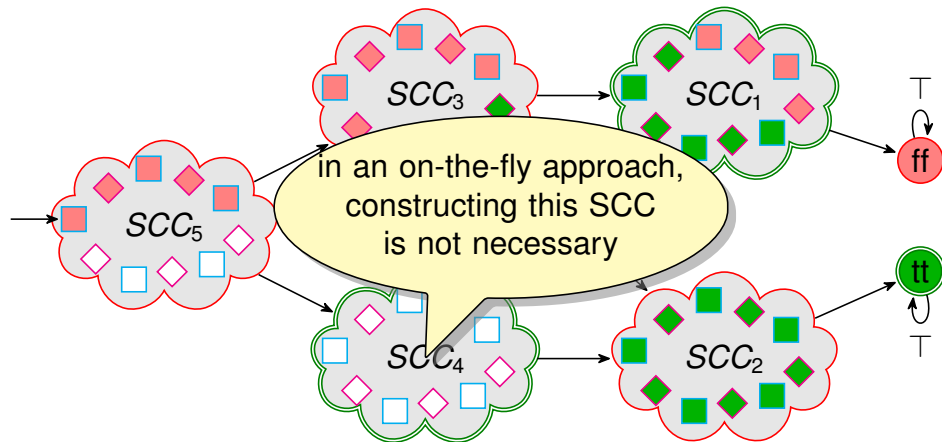
The **input player** wants the play to get stuck in a **rejecting SCC**.  
The **output player** wants the play to get stuck in an **accepting SCC**.



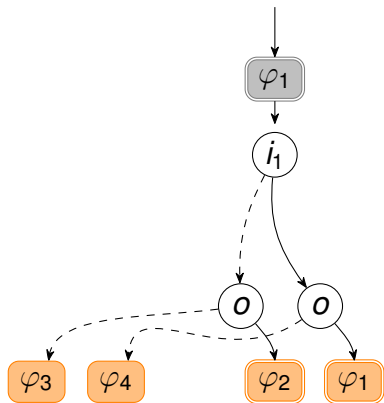
# Solving Weak Games Bottom-Up

The **input player** wants the play to get stuck in a **rejecting SCC**.

The **output player** wants the play to get stuck in an **accepting SCC**.



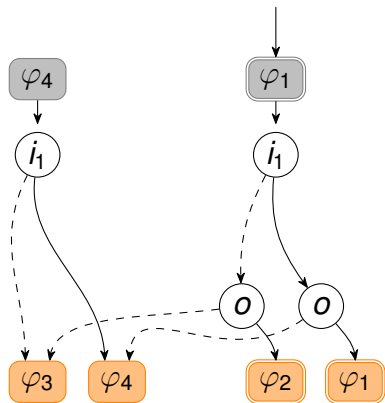
# On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1\}$

DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis

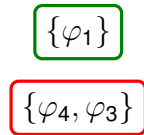
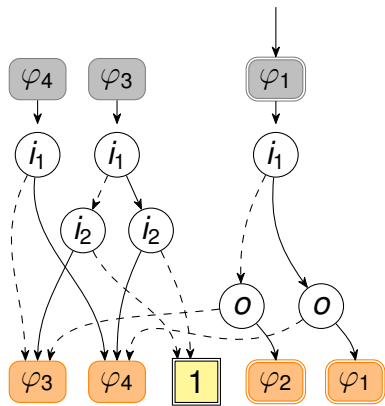


$\{\varphi_1\}$

$\{\varphi_4\}$

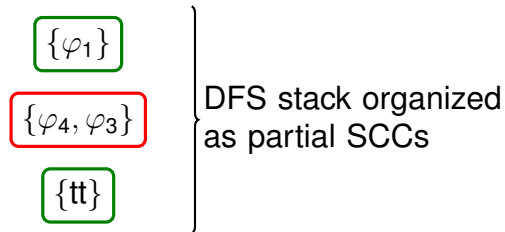
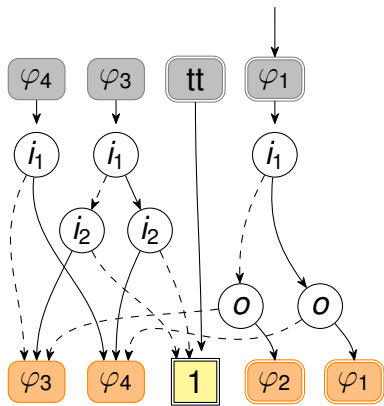
DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis

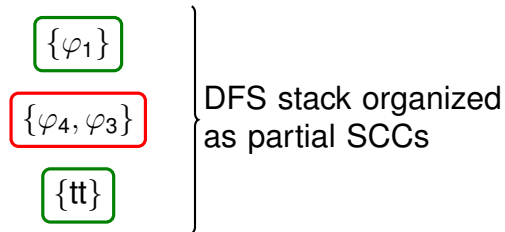
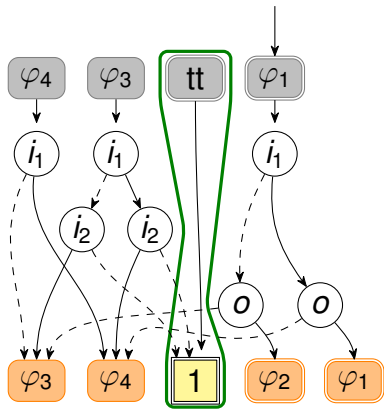


DFS stack organized  
as partial SCCs

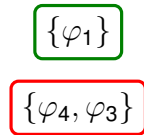
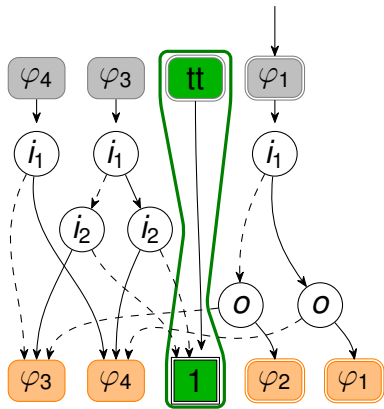
# On-the-Fly MTBDD-based Obligation Synthesis



# On-the-Fly MTBDD-based Obligation Synthesis

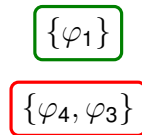
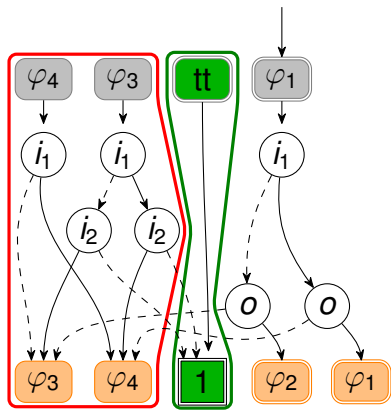


# On-the-Fly MTBDD-based Obligation Synthesis



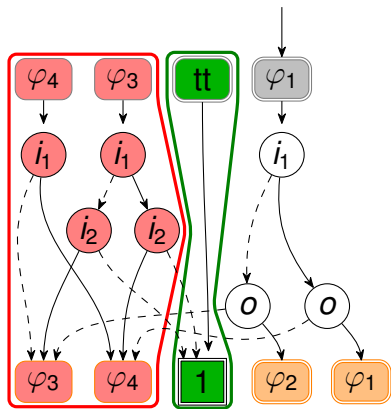
DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis



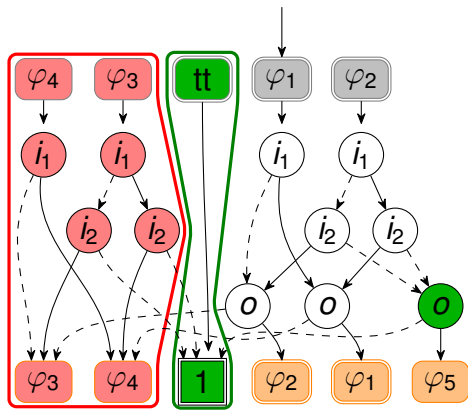
DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis


$$\{\varphi_1\}$$

- DFS stack organized as partial SCCs

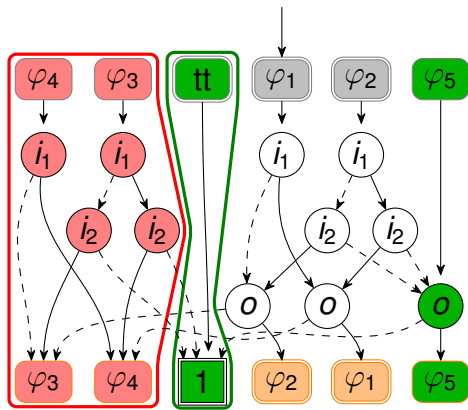
# On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis

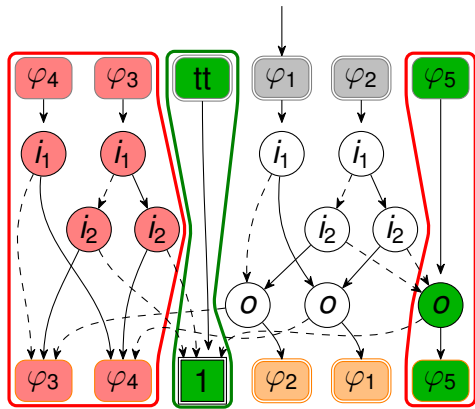


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized  
as partial SCCs

# On-the-Fly MTBDD-based Obligation Synthesis

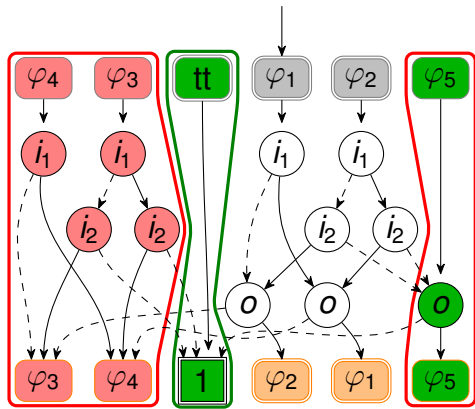


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized  
as partial SCCs

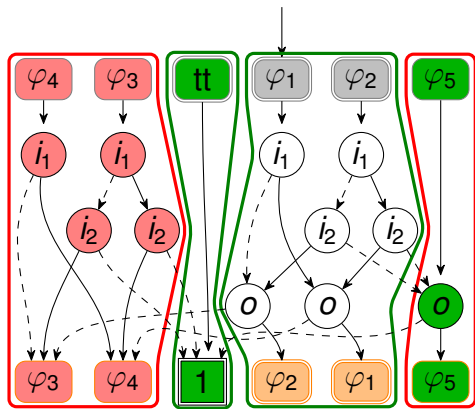
# On-the-Fly MTBDD-based Obligation Synthesis



$\{ \varphi_1, \varphi_2 \}$

DFS stack organized  
as partial SCCs

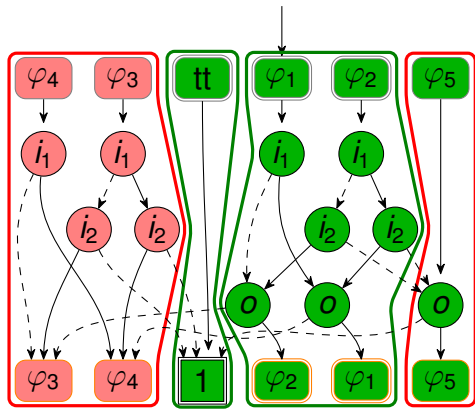
# On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized  
as partial SCCs

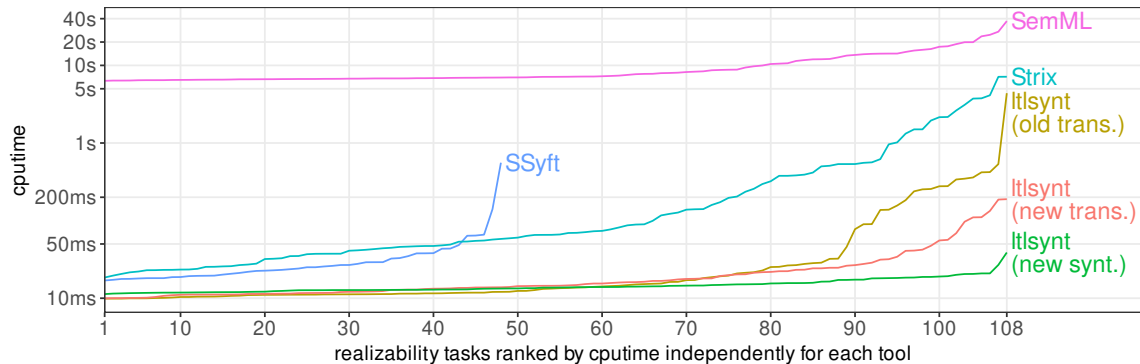
# On-the-Fly MTBDD-based Obligation Synthesis



DFS stack organized  
as partial SCCs

# Improving `ltlsynt` on Obligations

The SyntComp repository contains 108 obligations.



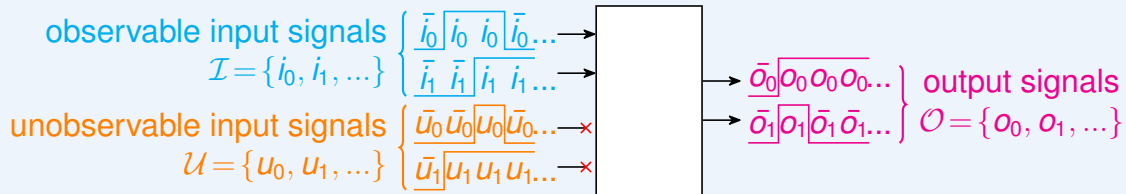
# Outline

- 1 LTL<sub>f</sub> Synthesis
- 2 Obligation Synthesis (LTL)
- 3 Synthesis Under Partial Observability
  - Introduction
  - An LTL<sub>f</sub> Example
  - Extensions



# Reactive Synthesis with Partial Observation

The reactive controller is **blind** to some of the inputs



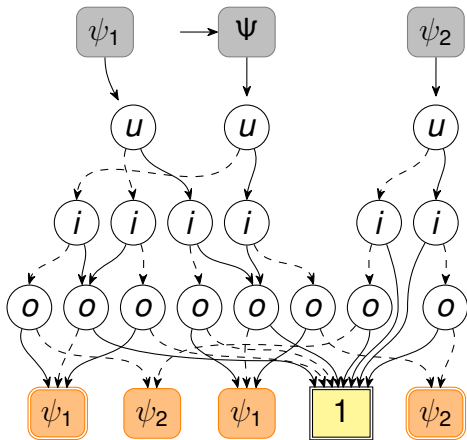
The realizability problem

Given a specification  $\varphi$  over  $\mathcal{I} \cup \mathcal{U} \cup \mathcal{O}$ , we want to know if  $\boxed{\forall \mathcal{U}, \varphi}$  is realizable.

An automaton for  $\boxed{\forall \mathcal{U}, \varphi}$  can be computed on-the-fly for  $\text{LTL}_f$  or for obligations!

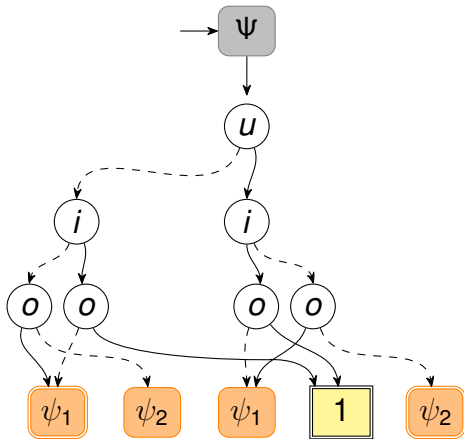
# An LTLf Example

$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



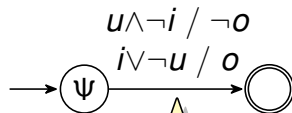
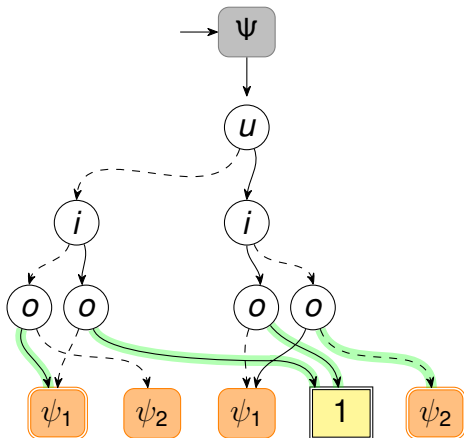
# An LTLf Example

$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



# An LTLf Example

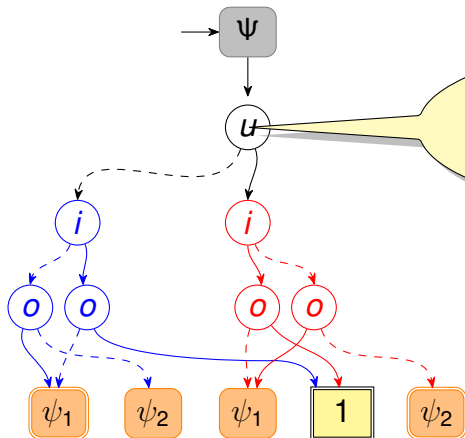
$$\psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



With full observability,  
 $\psi$  is realizable in one step.

# An LTLf Example

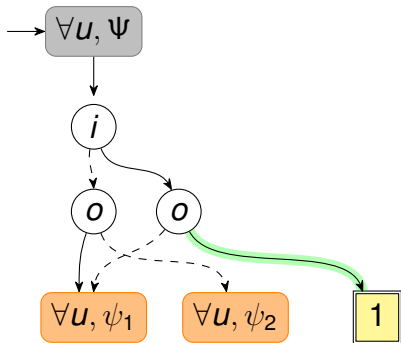
$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



To compute  $\forall u, \Psi$   
we use quantification on MTBDD:  
 $\forall u, B = B[u \leftarrow \perp] \wedge B[u \leftarrow \top]$

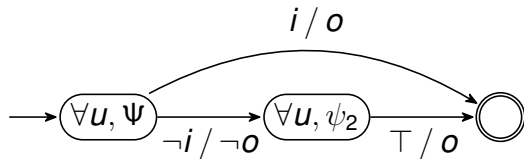
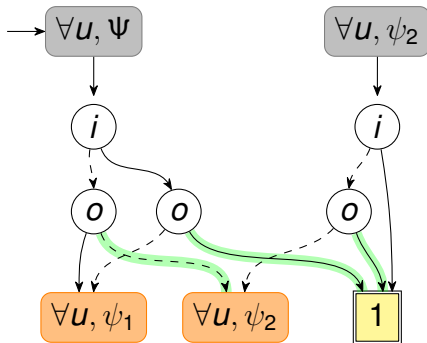
# An LTLf Example

$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



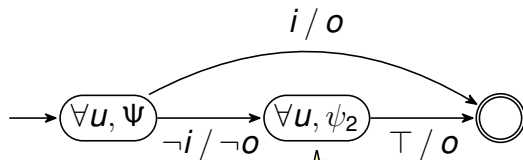
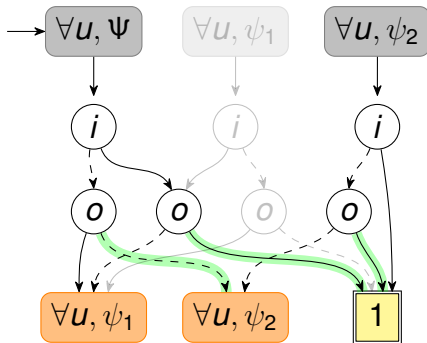
# An LTLf Example

$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



# An LTLf Example

$$\Psi = \underbrace{((GFu) \rightarrow F(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((GF\neg u) \rightarrow F(i \vee o))}_{\psi_2}$$



$\forall u, \Psi$  is not always realizable in one step

# Extensions

- ▶ Computing  $\forall \mathcal{U}, \varphi$  during translation using MTBDD works for:
  - ▶  $\text{LTL}_f \rightarrow \text{DFA}$
  - ▶  $\text{Obligation} \rightarrow \text{WDBA}$
- ▶ For these two translations  $\exists \mathcal{V}, \varphi$  can be supported similarly.

For full LTL, synthesis under partial observation can be done as follows:

- 1 compute a NBA for  $\exists \mathcal{U}, \neg \varphi$
- 2 complement NBA to DPA (i.e., a DPA for  $\forall \mathcal{U}, \varphi$ )
- 3 solve parity game

All of this is implemented in Spot 2.15



# Conclusion

We have discussed 3 MTBDD-based techniques for synthesis problems:

- 1  $LTL_f$  Synthesis
- 2 Obligation Synthesis (LTL)
- 3 Synthesis Under Partial Observability

Future work: generalize to full LTL.

-  Duret-Lutz et al. *Engineering an  $LTL_f$  synthesis tool*. **CIAA'25**. [▶ doi](#)
-  Duret-Lutz et al. *Fast obligation translation and synthesis*. **CAV'26**. [▶ doi](#)
-  Alon et al. *On-the-fly  $LTL_f$  synthesis under partial observability*. **KR'26**. [▶ doi](#)

## 1. Title

### 1 LTLf Synthesis

- 3. Reactive Synthesis
- 4. Text-Book Approach
- 5. Stopping on Final States
- 6. MTBDD/MTDFA
- 7. Outline
- 8. MTDFA as game
- 10.  $LTL_f \rightarrow MTBDD$  example
- 11.  $LTL_f \rightarrow MTBDD$  formal
- 12.  $LTL_f \rightarrow MTDFA$
- 13. Accepting Terminals
- 15. On-the-Fly
- 17. Benchmark
- 18. Conclusion

### 2 Obligation Synthesis

- 20. MTDBA
- 21. Hierarchy
- 22. Contributions
- 23. Obligations
- 24.  $\lambda$
- 25. Obligation  $\rightarrow$  MTBDD
- 27. Weak Games
- 28. On-the-fly
- 29. Speedup

### 3 Partial Observability

- 31. Intro
- 32. Example
- 33. Extensions

### 4 Extra

- 36. Preprocessings
- 37. Propositional Equivalence
- 38. Obligations Frequency
- 39. LTL Synthesis Pipeline

# Preprocessings

## Simplify specification using polarity of propositions

- ▶ If an **output** proposition is always positive/negative in the specification, replace it by  $\top/\perp$ .
- ▶ If an **input** proposition is always positive/negative in the specification, replace it by  $\perp/\top$ .

Example:  $G(i \rightarrow o)$  becomes  $G(\top \rightarrow \top) \equiv \top$ .

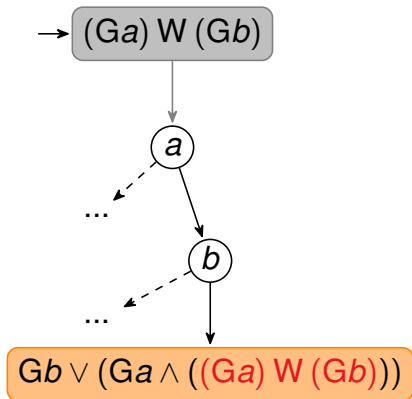
## Use cheap rewritings to reduce number of MTBDD operations

$$X\alpha \wedge X\beta \rightsquigarrow X(\alpha \wedge \beta), \quad (\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma) \rightsquigarrow \alpha \rightarrow (\beta \wedge \gamma), \quad \dots$$

## Split specification into output disjoint specifications when possible

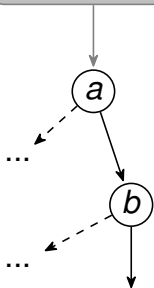
$\Psi_1 \wedge \Psi_2$  can be solved as two independent problems if  $\Psi_1$  and  $\Psi_2$  are output-disjoint, and admit controllers that agree on accepting lengths.

# Propositional Equivalence



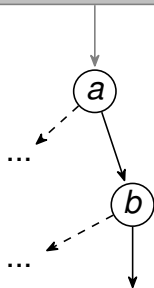
# Propositional Equivalence

→  $(Ga) \vee (Gb)$



$Gb \vee (Ga \wedge ((Ga) \vee (Gb)))$

$Gb \vee (Ga \wedge ((Ga) \vee (Gb)))$

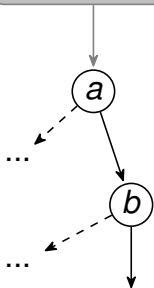


$Gb \vee (Ga \wedge (Gb \vee (Ga \wedge ((Ga) \vee (Gb)))))$

...

# Propositional Equivalence

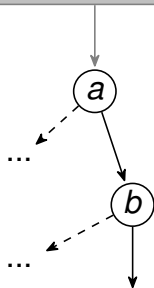
→  $(Ga) \vee (Gb)$



$Gb \vee (Ga \wedge ((Ga) \vee (Gb)))$

$p_3 \vee (p_2 \wedge p_1)$

$Gb \vee (Ga \wedge ((Ga) \vee (Gb)))$

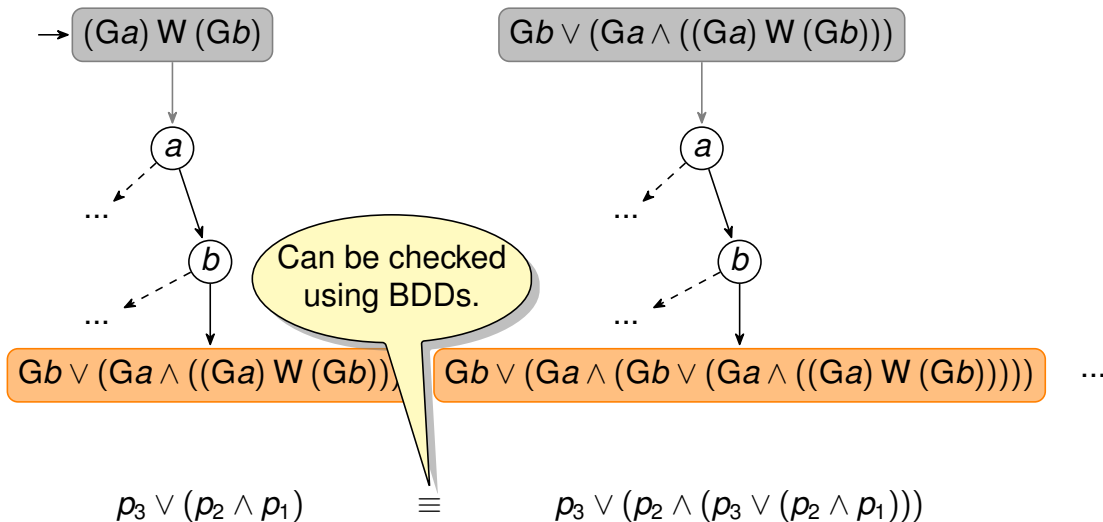


$Gb \vee (Ga \wedge (Gb \vee (Ga \wedge ((Ga) \vee (Gb)))))$

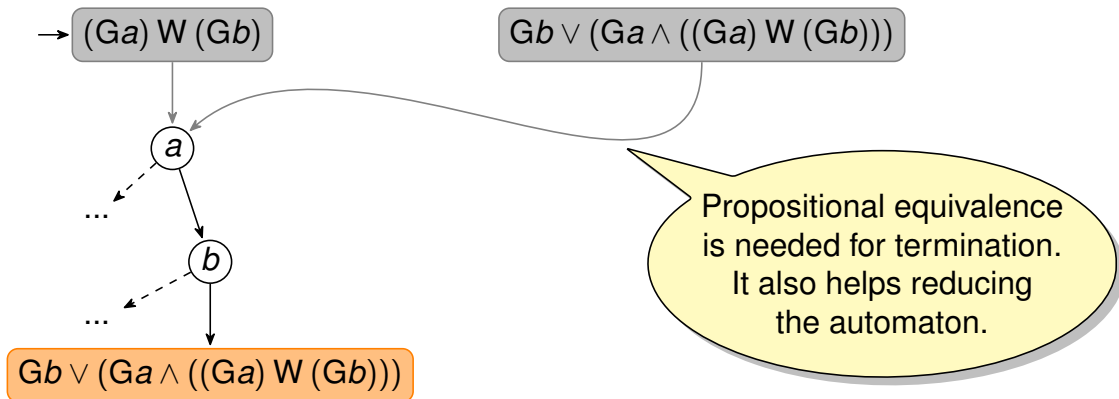
$p_3 \vee (p_2 \wedge (p_3 \vee (p_2 \wedge p_1)))$

...

# Propositional Equivalence



# Propositional Equivalence



# Obligations Frequency

| source            | set size | property classes |                             |                 |                 |          | syntactic classes |                             |                 |                 |     |
|-------------------|----------|------------------|-----------------------------|-----------------|-----------------|----------|-------------------|-----------------------------|-----------------|-----------------|-----|
|                   |          | $O$              | $O \setminus S \setminus G$ | $S \setminus B$ | $G \setminus B$ | $B$      | $O$               | $O \setminus S \setminus G$ | $S \setminus B$ | $G \setminus B$ | $B$ |
| SyntComp          | 959      | $\geq 262$       | $\geq 172$                  | $\geq 73$       | $\geq 7$        | $\geq 7$ | 110               | 59                          | 48              | 1               | 2   |
| Dwyer et al.      | 55       | 40               | 2                           | 37              | 1               |          | 25                | 13                          | 11              | 1               |     |
| Somenzi&Bloem     | 27       | 16               | 2                           | 6               | 6               | 2        | 13                | 4                           | 4               | 5               |     |
| Etessami&Holzmann | 12       | 5                |                             |                 | 4               | 1        | 5                 |                             |                 | 5               |     |
| BEEM              | 20       | 10               | 1                           | 6               | 1               | 2        | 7                 | 2                           | 4               | 1               |     |
| Liberouter        | 55       | 30               |                             | 27              | 1               | 2        | 26                | 1                           | 23              | 2               |     |

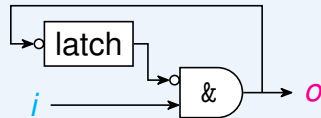
-  Jacobs et al. *The reactive synthesis competition (SYNTCOMP): 2018–2021*. [arXiv, 2022](#). [doi](#)
-  Dwyer, Avrunin, and Corbett. *Property specification patterns for finite-state verification*. **FMSP'98**. [doi](#)
-  Somenzi and Bloem. *Efficient Büchi automata for LTL formulae*. **CAV'00**. [doi](#)
-  Etessami and Holzmann. *Optimizing Büchi automata*. **Concur'00**. [doi](#)
-  Pelánek. *BEEM: benchmarks for explicit model checkers*. **Spin'07**. [doi](#)
-  Holeček et al. *Verification results in Liberouter project*. **Technical Report 03, CESNET, 2004**

# Textbook Reactive Synthesis

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

## 6. Output ALG



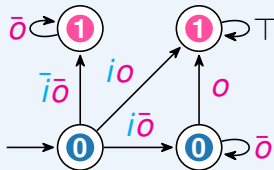
# Textbook Reactive Synthesis

## 1. LTL Spec.

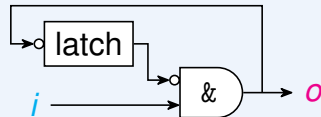
$$i \leftrightarrow F(o)$$

## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 6. Output ALG



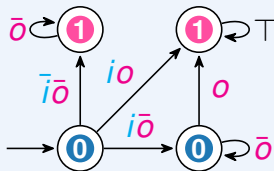
# Textbook Reactive Synthesis

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

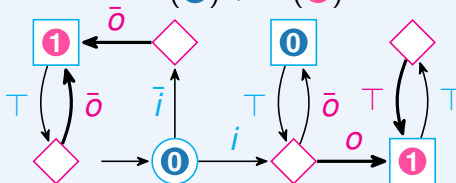
## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$

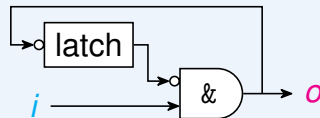


## 3. Parity Game (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 6. Output ALG



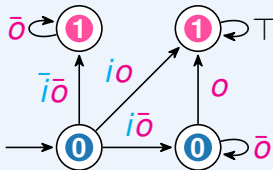
# Textbook Reactive Synthesis

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

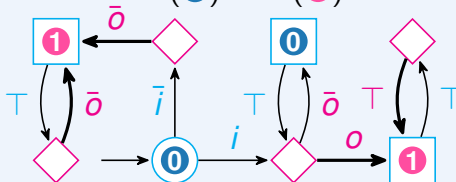
## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$

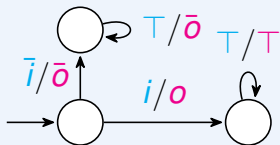


## 3. Parity Game (max odd)

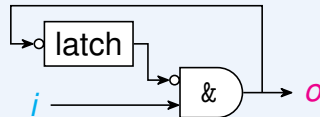
$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 4. Winning Strategy



## 6. Output ALG



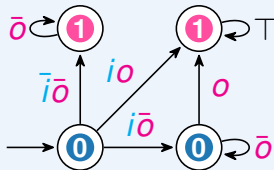
# Textbook Reactive Synthesis

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

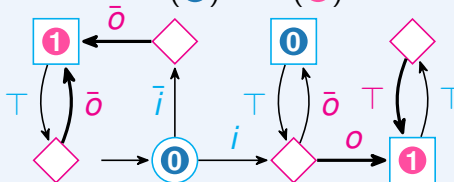
## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$

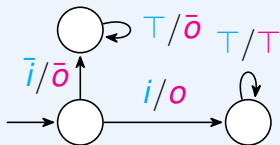


## 3. Parity Game (max odd)

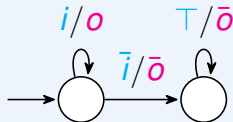
$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



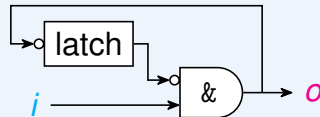
## 4. Winning Strategy



## 5. Simplified Strategy



## 6. Output ALG



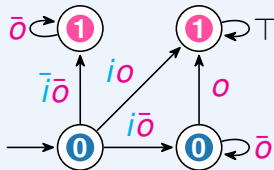
# Textbook Reactive Realizability

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

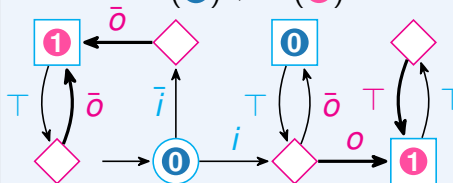
## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 3. Parity Game (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 4. Result

REALIZABLE

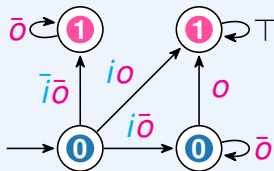
# Textbook Reactive Realizability

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

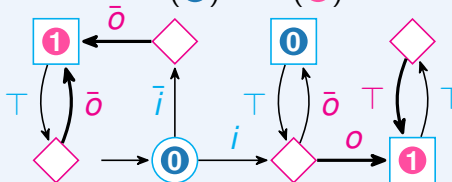
## 2. DPA (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 3. Parity Game (max odd)

$$\text{Fin}(\textcircled{0}) \vee \text{Inf}(\textcircled{1})$$



## 4. Result

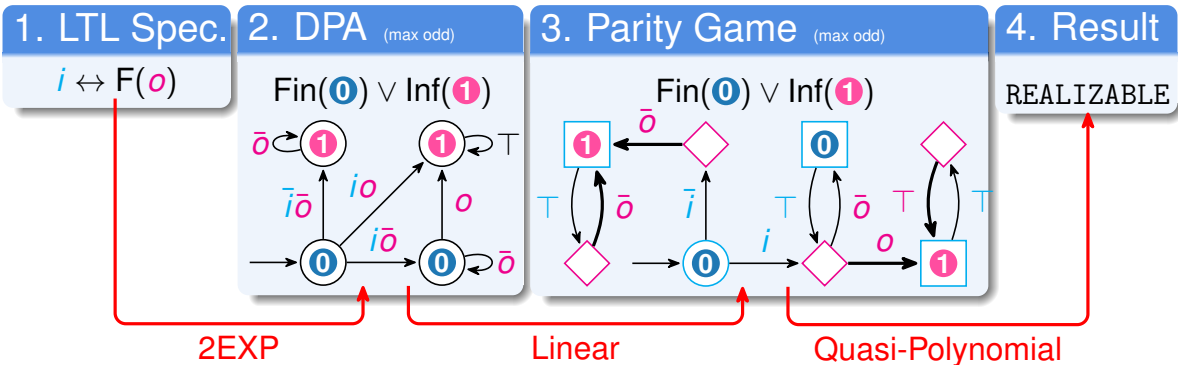
REALIZABLE

2EXP

Linear

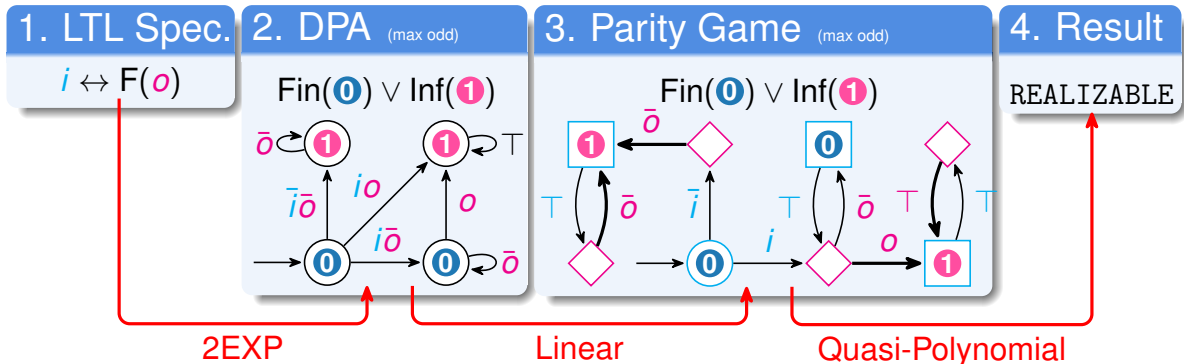
Quasi-Polynomial

# Textbook Reactive Realizability



Contribution: faster pipeline for the “syntactic obligation” fragment of LTL.

# Reactive Realizability



Contribution: faster pipeline for the “syntactic obligation” fragment of LTL.

MTBDD-based repr. of Weak DBA,  
usable directly as a game

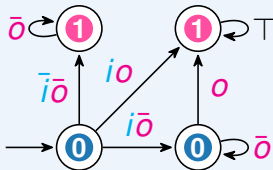
# Reactive Realizability

## 1. LTL Spec.

$$i \leftrightarrow F(o)$$

## 2. DPA (max odd)

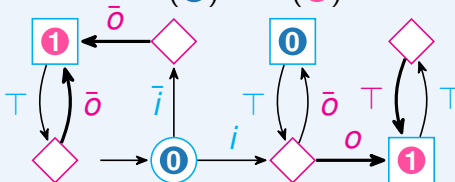
$$\text{Fin}(\mathbf{0}) \vee \text{Inf}(\mathbf{1})$$



2EXP

## 3. Parity Game (max odd)

$$\text{Fin}(\mathbf{0}) \vee \text{Inf}(\mathbf{1})$$



Linear

Quasi-Polynomial

## 4. Result

REALIZABLE

Contribution: faster pipeline for the “syntactic obligation” fragment of LTL.

MTBDD-based repr. of Weak DBA,  
usable directly as a game

2EXP

(but simpler, faster)

Linear

(weak game solving)